İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ
2010
FEN BİLİMLERİ ENSTİTÜSÜ

Repeticards, Kategorilere Dayalı Flaş Kartlarla Aralıklı Tekrarlama Tekniğini Kullanarak İngilizce Kelime Öğrenme iOS Uygulaması

Yazılım Mühendisliği Ana Bilim Dalı

Dönem Projesi

Pakize Selek

Proje Danışmanı: Dr. Öğr. Üyesi Serpil Yılmaz

Ocak 2024

Repeticards, Kategorilere Dayalı Flaş Kartlarla Aralıklı Tekrarlama Tekniğini Kullanarak İngilizce Kelime Öğrenme iOS Uygulaması

Öz

Repeticards, kategorilere dayalı flaş kartlarla ve aralıklı tekrarlama tekniğini kullanarak İngilizce kelime öğrenmeyi kolaylaştırmak için tasarlanmış yenilikçi bir iOS uygulamasıdır. Bu uygulama, aralıklı tekrarlamanın bilişsel faydalarını kullanarak İngilizce kelimelerin hafızada tutulmasını ve hatırlanmasını güçlendirir. Kullanıcılar yeni kelimeler ekleyebilir, bunları kategorize edebilir ve kişiselleştirilmiş programlara göre sistematik bir şekilde gözden geçirebilirler. Uygulama, öğrenme deneyimini zenginleştirmek için görsel ve sesli unsurlar içerir. Bu akademik makale, Repeticards'ın geliştirme sürecini, altında yatan eğitim teorilerini, teknik mimarisini ve kullanıcı arayüzü tasarımını detaylandırarak, etkili ve çekici bir dil edinimi aracı olarak sunmaktadır.

Anahtar Sözcükler: Aralıklı tekrarlama, İngilizce kelime öğrenimi, iOS uygulaması, flaş kartlar, dil edinimi, eğitimsel teknoloji, bilişsel öğrenme, kullanıcı arayüzü tasarımı, çoklu medya öğrenimi, kişiselleştirilmiş öğrenme programları

Repeticards, Word Learning iOS Application Using Spaced Repetition Technique with Category-Based Flashcards

Abstract

Repeticards is an innovative iOS application designed to facilitate English vocabulary learning through the spaced repetition technique, utilizing categorized flashcards. This application leverages the cognitive benefits of spaced repetition, enhancing memory retention and recall of English words. Users can add new words, categorize them, and review them systematically according to personalized schedules. The app incorporates multimedia elements such as images and audio to enrich the learning experience. This academic paper details the development process, underlying educational theories, technical architecture, and user interface design of Repeticards, presenting it as a comprehensive tool for effective and engaging language acquisition.

Keywords: Spaced repetition, English vocabulary learning, iOS application, flashcards, language acquisition, educational technology, cognitive learning, user interface design, multimedia learning, personalized learning schedules

Bu projemi çalışmamı, hayatımın her aşamasında ve eğitim öğretim hayatım boyunca bana her zaman destek olan ve her zaman yanımda olan, çalışma hayatım boyunca bana her zaman destek olan ve her zaman yanımda olan, eğitim öğretime; ilime ve bilime çok önem veren, ilim tahsil eden insanları çok seven, yetiştirdiğim her öğrencide kendisinin imzası olan, ileri görüşlü, aydın ve çalışkan olan, beni dünyalar kadar çok seven, yakın zaman önce kaybettiğim canım babacığım rahmetli Şinasi YÜNDEN'e ithaf ediyorum.

Teşekkür

Proje çalışmasına katkılarından dolayı eşim Eyüp Harun Selek'e; danışman hocam sayın Serpil Yılmaz'a çok teşekkür ediyorum.

İçindekiler

Şekiller Listesi

1. Giriş

Günümüzde, teknolojinin hızla ilerlemesi ve bilgiye erişim olanaklarının artmasıyla birlikte, yabancı dil öğrenme süreci de önemli ölçüde dönüşüm geçirmiştir. İngilizce, globalleşen dünyamızda en çok talep edilen dillerden biri olarak öne çıkmakta ve bu dilin öğrenilmesi bireysel ve profesyonel başarının önemli bir parçası haline gelmiştir. Bu bağlamda, etkili öğrenme yöntemleri ve teknolojik araçların kullanımı, dil öğrenme sürecini optimize etme açısından kritik öneme sahiptir.

"Repeticards" isimli bu projemiz, İngilizce kelime öğrenimini desteklemek amacıyla tasarlanmış bir iOS uygulamasıdır. Uygulama, aralıklı tekrarlama tekniğine dayalı, kategorilere ayrılmış flaş kartlar kullanarak kullanıcıların yeni kelimeleri daha etkin ve kalıcı bir şekilde öğrenmelerini amaçlamaktadır. Aralıklı tekrarlama, bilgiyi uzun süreli hafızaya aktarmanın ve pekiştirmenin kanıtlanmış bir yöntemidir. Bu projede, bu yöntemin mobil teknoloji ile entegrasyonu ve uygulama içi interaktif özelliklerle zenginleştirilmesi hedeflenmiştir.

Projemiz, kullanıcıların İngilizce kelime dağarcıklarını genişletmelerine yardımcı olmayı, öğrenme sürecini kişiselleştirmeyi ve daha interaktif hale getirmeyi amaçlamaktadır. Bu hedeflere ulaşmak için uygulama, kullanıcı dostu bir arayüz, görsel ve işitsel öğrenme unsurları ile zenginleştirilmiş içerikler ve kişisel öğrenme ihtiyaçlarına uygun esnek bir öğrenme programı sunmaktadır.

Bu tez çalışması, "Repeticards" uygulamasının tasarımı, geliştirilme süreci, kullanılan teknolojik araçlar ve eğitim teorileri üzerine odaklanacak, uygulamanın dil öğrenimindeki etkinliğini ve potansiyelini değerlendirecektir. Ayrıca, kullanıcı deneyimleri ve geri bildirimleri, uygulamanın sürekli iyileştirilmesi ve geliştirilmesi için temel bir kaynak olarak ele alınacaktır.

2. Kaynak Araştırması

Bu çalışmada ele alınan "Repeticards" uygulaması, İngilizce kelime öğreniminde aralıklı tekrarlama tekniğinin ve teknolojik araçların kullanımının etkinliğini vurgulamaktadır. Aralıklı tekrarlama, bilginin zamanla tekrar edilerek uzun süreli hafızada tutulmasını sağlayan bir öğrenme tekniğidir. Ebbinghaus'un unutma eğrisi teorisi (1885), bu teknikle ilgili erken çalışmalardan biridir ve bilginin zaman içinde nasıl unutulduğunu ve düzenli tekrarın hafızayı nasıl güçlendirdiğini ortaya koymuştur.

Teknolojinin eğitimde kullanımı ile ilgili literatür, çeşitli araştırmaları içermektedir. Mayer (2001) ve Clark & Mayer (2016) tarafından geliştirilen Çoklu Ortam Öğrenme Teorisi, görsel ve işitsel materyallerin öğrenme sürecine entegrasyonunun önemini vurgular. Bu teori, RepetiCards'ın tasarımında, kelime ve anlamlarının görsel ve işitsel unsurlarla zenginleştirilmesinde önemli bir rol oynamaktadır.

Mobil öğrenme (m-learning) alanındaki çalışmalar da bu projenin temelini oluşturur. Crompton (2013) ve Kukulska-Hulme (2010), mobil cihazların öğrenme sürecindeki esnekliğini ve erişilebilirliğini vurgulayan çalışmalar yapmışlardır. Repeticards, bu esneklik ve erişilebilirlik prensiplerini benimseyerek kullanıcılara her yerde öğrenme imkanı sunar.

Ayrıca, aralıklı tekrarlama tekniğinin yabancı dil öğrenimine etkisi üzerine yapılan çalışmalar da projenin bilimsel temelini güçlendirmektedir. Wozniak (1990) ve Kapler et al. (2015) tarafından yapılan araştırmalar, bu tekniğin kelime hatırlama başarısını önemli ölçüde artırdığını göstermektedir.

Son olarak, kullanıcı arayüzü tasarımı ve kullanıcı deneyimi (UX) konusundaki literatür, Repeticards'ın kullanıcı dostu ve etkileşimli arayüzünün geliştirilmesinde önemli bir kaynak olarak hizmet etmiştir. Nielsen ve Norman (2000) tarafından geliştirilen kullanıcı deneyimi ilkeleri, bu alandaki temel rehberlerden biridir.

Bu kaynak araştırması, Repeticards uygulamasının bilimsel ve pratik temellerini açıkça ortaya koymaktadır ve uygulamanın geliştirilmesindeki teorik yaklaşımları desteklemektedir.

3. Materyal ve Yöntem

Bu çalışmanın odak noktası olan Repeticards uygulaması, İngilizce kelime öğrenimini desteklemek için tasarlanmış bir iOS mobil uygulamasıdır. Uygulamanın geliştirilmesi sürecinde çeşitli yazılım geliştirme araçları ve metodolojileri kullanılmıştır.

3.1. Mobil Uygulama

Mobil uygulama bölümünde uygulama geliştirilirken yazılan kodlar detaylı bir şekilde anlatılacaktır. RepetiCards, Apple'ın native uygulama geliştirme platformu olan Swift ve SwiftUI kullanılarak XCode ile geliştirilmiştir. SwiftUI, modern bir kullanıcı arayüzü tasarımı için gerekli araçları sağlamaktadır. Uygulama pek çok dosyadan oluşmaktadır. Aşağıda bu dosyaları teker teker inceleyeceğiz.

3.1.1. RepetiCardsApp.swift (kaynak kod EK 1'de görülebilir)

"RepetiCardsApp.swift" dosyası, RepetiCards uygulamasının temel taşıdır ve tüm uygulamanın başlangıç noktasını oluşturur. Swift ve SwiftUI'nin sunduğu modern özellikler sayesinde, bu dosya uygulamanın ana yapısını, başlangıç ayarlarını ve ana giriş noktasını tanımlar. Uygulamanın yaşam döngüsünü ve ana ekranını yönetmekle sorumludur. Bu dosya, SwiftUI'nin App protokolünü uygulayan ve @main etiketi ile işaretlenen bir struct içerir. Bu etiket, uygulamanın başlangıç noktasını belirtir ve Swift'in programın nereden başlayacağını anlamasını sağlar.

3.1.1.1. CoreDataStack ve ViewModel Entegrasyonu

Dosya içerisinde, uygulamanın veri yönetimine ilişkin önemli bir bileşen olan CoreDataStack.shared örneği yer alır. Bu, veri modellemesi ve yerel veri depolaması için kullanılan Core Data'nın başlatılmasını ve yönetimini sağlar. Bu sayede uygulamanın veri tabanı işlemleri, güvenilir ve etkin bir şekilde gerçekleştirilir. Ayrıca, dosyada FlashcardViewModel örneği(instance) de başlatılır. Bu ViewModel, uygulamanın veri ve iş mantığını içerir ve görünüm katmanı ile veri tabanı katmanı arasındaki etkileşimi sağlar. ViewModel'in bu dosyada başlatılması, uygulamanın tüm görünümlerinde veri tutarlılığını ve senkronizasyonunu garanti eder.

3.1.1.2. Bildirim Yönetimi ve Kullanıcı İzni

Uygulamanın başlatılması sırasında, NotificationManager.shared.requestAuthorization() metodu çağrılır. Bu, uygulamanın kullanıcıdan bildirim gönderme izni istemesini sağlar. Bildirim yönetimi, özellikle kelime öğrenme uygulamalarında kritik bir rol oynar çünkü kullanıcıların öğrenme süreçlerini desteklemek ve onları düzenli tekrara teşvik etmek için etkili bir yöntemdir. Bu izin, kullanıcının uygulama içi deneyimini kişiselleştirmesine ve öğrenme hatırlatıcıları almasına olanak tanır.

3.1.1.3. Uygulama Arayüzü ve Navigasyon

Dosya ayrıca uygulamanın ana arayüzünü tanımlar. WindowGroup içindeki TabView, uygulamanın tab tabanlı ana yapısını oluşturur ve kullanıcıya farklı ekranlar arasında geçiş yapma imkanı sunar. Burada, ContentView ve AddWordView gibi alt görünümler tab item olarak eklenir ve her birine özel etiketler atanır. Bu yapı, kullanıcının uygulama içinde rahatça gezinmesini sağlar ve farklı özelliklere kolay erişim imkanı sunar.

3.1.2. CoreDataStack.swift (kaynak kod EK 2'de görülebilir)

"CoreDataStack.swift" dosyası, RepetiCards uygulamasının veri yönetimini ve saklanmasını merkezi olarak ele alan bir yapılandırmadır. Bu dosya, Apple'ın Core Data framework'ünü kullanarak, uygulamanın tüm veri modellemesi ve yerel veri depolama işlemlerini yönetir. Core Data, nesne grafiği yönetimi ve veritabanı entegrasyonu için güçlü ve verimli bir çözüm sunar, bu sayede uygulamanın veri tabanı işlemleri daha organize ve sürdürülebilir hale gelir.

3.1.2.1. Core Data Stack Yapılandırması

Dosya, Core Data stack'ını başlatan ve yapılandıran CoreDataStack adında bir sınıf içerir. Bu sınıf, uygulamanın veri modelini temsil eden NSPersistentContainer'ı tanımlar ve başlatır. NSPersistentContainer, uygulamanın veri modeli olan "RepetiCards" ile ilişkilendirilmiş ve Core Data model dosyasının (.xcdatamodeld) adı ile eşleştirilmiştir. Container, veritabanı bağlantısını yönetir ve uygulamanın veri tabanı işlemleri için gerekli olan kontekstleri sağlar.

3.1.2.2. Context Yönetimi ve Veri Kaydetme İşlemleri

Dosyanın içinde, uygulamanın ana işlem konteksti olan context özelliği bulunur. Bu, veritabanıyla etkileşimde bulunmak için kullanılan ana arayüzdür. saveContext metodu, yapılan değişiklikleri veritabanına kaydetmek için kullanılır. Bu metod, veri tabanında yapılan değişikliklerin güvenli ve tutarlı bir şekilde kaydedilmesini sağlar. Eğer kaydetme sırasında bir hata oluşursa, bu hata yakalanır ve uygulama geliştiricisine bilgi verilir, bu sayede veri kaybı önlenmiş olur.

3.1.2.3. Veri Bütünlüğü ve Güvenliği

CoreDataStack sınıfı, veri bütünlüğünü korumak ve olası hataları yönetmek için dikkatli bir şekilde tasarlanmıştır. Veri kaydetme işlemleri, uygulamanın stabilitesini ve güvenliğini sağlamak için dikkatlice ele alınmıştır. Bu yaklaşım, uygulamanın veri tabanıyla etkileşimlerinde güvenilirlik ve verimlilik sağlar.

3.1.3. ContentView.swift (kaynak kod EK 3'de görülebilir)

"ContentView.swift" dosyası, RepetiCards uygulamasının ana kullanıcı arayüzünü tanımlar ve uygulamanın görsel yüzünün temelini oluşturur. SwiftUI framework'ünün getirdiği deklaratif UI yapısı sayesinde, bu dosya, uygulamanın kullanıcıya ilk görünen ekranını ve kullanıcı etkileşimlerinin başlangıç noktasını oluşturur. Bu dosya, kullanıcıya uygulamanın ana özelliklerine erişim sağlayan bir navigasyon ve düzen sunar.

3.1.3.1. Navigasyon ve Görünüm Yapısı

Dosyanın içinde tanımlanan ContentView struct'ı, bir NavigationView içerir. Bu, kullanıcının uygulama içinde gezinmesini kolaylaştıran ve içeriği düzenleyen bir SwiftUI elementidir. NavigationView, kullanıcıya diğer alt görünümlere erişim imkanı sunar ve uygulamanın farklı bölümleri arasında geçiş yapmayı sağlar. Dosyada özellikle CategoryListView gibi alt görünümlere yönlendirme yapılır, bu da uygulamanın modüler ve genişletilebilir bir yapıda olmasını sağlar.

3.1.3.2. Kullanıcı Arayüzü Tasarımı

ContentView içerisindeki kullanıcı arayüzü elemanları, SwiftUI'nin güçlü ve esnek UI bileşenleri kullanılarak tasarlanmıştır. Bu bileşenler, uygulamanın estetik ve kullanıcı

dostu bir arayüz sunmasını sağlar. Arayüz, kullanıcının ihtiyaçlarını ve beklentilerini ön planda tutacak şekilde tasarlanmıştır, bu sayede kullanıcılar uygulamayı kolayca anlayabilir ve etkileşimde bulunabilir.

3.1.3.3. Veri Bağlama ve Güncellemeler

Dosyada ayrıca, FlashcardViewModel örneği kullanılarak veri bağlama işlemleri gerçekleştirilir. Bu, uygulamanın veri modeli ile kullanıcı arayüzü arasında dinamik bir bağlantı kurar ve veri değişikliklerinin gerçek zamanlı olarak kullanıcı arayüzüne yansıtılmasını sağlar. Bu veri akışı, uygulamanın reaktif ve interaktif bir kullanıcı deneyimi sunmasına olanak tanır.

3.1.4. FlashcardRepository.swift (kaynak kod EK 4'de görülebilir)

FlashcardRepository.swift" dosyası, RepetiCards uygulamasının veri tabanı işlemlerinin merkezini oluşturur. Bu dosya, uygulamanın veri tabanı ile olan tüm etkileşimlerini yönetir ve bu etkileşimler için bir arayüz sağlar. Core Data framework'ünü kullanarak, uygulamanın veri modeline erişim ve bu veriler üzerinde işlemler yapma yeteneğini sunar. Bu dosya, veri tabanı işlemlerini sapsamak, veri eklemek, güncellemek ve sorgulamak için gerekli metodları içerir.

3.1.4.1. Veri Tabanı İşlemleri ve Yönetimi

Dosya içinde tanımlanan FlashcardRepository sınıfı, uygulamanın veri tabanı ile ilgili işlemleri gerçekleştirir. Bu sınıf, Core Data'nın NSManagedObjectContext'ini kullanarak veri eklemek, güncellemek ve sorgulamak için metodlar sağlar. Örneğin, createCategory

metodu, yeni bir kategori oluşturmak, getAllWords metodu, kaydedilen tüm kelimeleri almak, addMeaning metodu ise bir kelimeye yeni bir anlam eklemek için kullanılır. Bu metodlar, veri tabanında verimli ve güvenli bir şekilde işlem yapılmasını sağlar.

3.1.4.2. Kategori ve Kelime Yönetimi

Dosyada, kategori ve kelime yönetimi ile ilgili özel işlevler bulunur. Bu işlevler, uygulamanın temelini oluşturan kelime ve kategori verilerini yönetmek için tasarlanmıştır. Örneğin, kelimeleri kategorilere atama, belirli bir kelimenin anlamlarını getirme gibi işlemler bu dosya aracılığıyla gerçekleştirilir. Bu yaklaşım, uygulamanın veri tabanı işlemlerinin modüler ve organize bir şekilde yönetilmesini sağlar.

3.1.4.3. Veri Bütünlüğü ve Güvenliği

Dosyadaki saveContext ve saveChanges metodları, veri tabanındaki değişikliklerin güvenli bir şekilde kaydedilmesini sağlar. Bu metodlar, uygulamanın veri bütünlüğünü korur ve olası hataları yönetir. Veri kaydetme işlemleri sırasında oluşabilecek hatalar, uygulama geliştiricisine bildirilir ve bu sayede veri kaybı veya tutarsızlıkların önüne geçilir.

3.1.5. FlashcardViewModel.swift (kaynak kod EK 5'de görülebilir)

"FlashcardViewModel.swift" dosyası, RepetiCards uygulamasının veri ve iş mantığının merkezini oluşturur. Bu dosya, uygulamanın veri tabanı işlemleri ile kullanıcı arayüzü arasında aracı görevi görür ve MVVM (Model-View-ViewModel) tasarım deseninin bir parçası olarak işlev görür. ViewModel, uygulamanın veri tabanı katmanından gelen verileri işler ve bu verileri kullanıcı arayüzüne sunmak için hazırlar. Ayrıca, kullanıcı etkileşimlerini alır ve bu etkileşimlere göre veri tabanı işlemlerini tetikler.

3.1.5.1. Veri Yönetimi ve İşlevselliği

Dosya içinde tanımlanan FlashcardViewModel sınıfı, uygulamanın çeşitli veri yönetimi işlemlerini gerçekleştirir. Örneğin, kelime listelerini yüklemek, kategorilere göre düzenlemek, ve kullanıcı tarafından eklenen yeni kelimeleri ve anlamlarını işlemek gibi görevler bu sınıf tarafından yürütülür. @Published özellikleri sayesinde, bu sınıfın değişkenleri SwiftUI'nin veri bağlama mekanizmasıyla uyumlu hale gelir ve kullanıcı arayüzündeki herhangi bir değişiklik otomatik olarak güncellenir.

3.1.5.2. Kullanıcı Etkileşimleri ve Güncellemeler

ViewModel, kullanıcı etkileşimlerine yanıt vermek için metodlar içerir. Örneğin, bir kelime veya anlamı gözden geçirme, yeni bir kelime eklemek veya bir kategoriyi güncellemek gibi işlevler, kullanıcı arayüzünden gelen taleplere göre tetiklenir. Bu etkileşimler, veri tabanı katmanı ile senkronize bir şekilde çalışır ve kullanıcının eylemlerine göre dinamik olarak güncellenir.

3.1.5.3. Aralıklı Tekrarlama ve Bildirim Yönetimi

Dosyada, aralıklı tekrarlama tekniğini destekleyen ve kullanıcılara özelleştirilmiş öğrenme deneyimi sunan işlevler bulunur. scheduleNextReview metodu, kelimenin bir sonraki gözden geçirme tarihini hesaplar ve bu tarihe göre kullanıcıya bildirim gönderilmesini sağlar. Bu, kullanıcının öğrenme sürecini destekler ve daha etkili bir kelime hatırlama sağlar.

3.1.6. WordListView.swift (kaynak kod EK 6'da görülebilir)

"WordListView.swift" dosyası, RepetiCards uygulamasında kullanıcıların kaydedilen tüm kelimeleri görebilecekleri bir arayüzü tanımlar. Bu dosya, uygulamanın kelime gözden geçirme ve kelime listeleme özelliklerinin bir parçası olarak işlev görür. Kullanıcıların kaydettikleri kelimeleri görmelerini, incelemelerini ve detaylarına erişmelerini sağlayan bir liste görünümü sunar.

3.1.6.1. Listeleme ve Navigasyon

Dosya içindeki WordListView struct'ı, bir List kullanarak uygulamanın FlashcardViewModel'inden alınan kelimeleri görüntüler. Her kelime, bir NavigationLink içinde sunulur. Bu, kullanıcıların her bir kelimenin detaylı anlamlarını ve örneklerini görebilecekleri MeaningListView gibi daha detaylı bir görünüme yönlendirilmesine olanak tanır. Bu yapı, kullanıcıların uygulama içinde rahatça gezinmelerini ve öğrenme materyallerine kolay erişim sağlamalarını destekler.

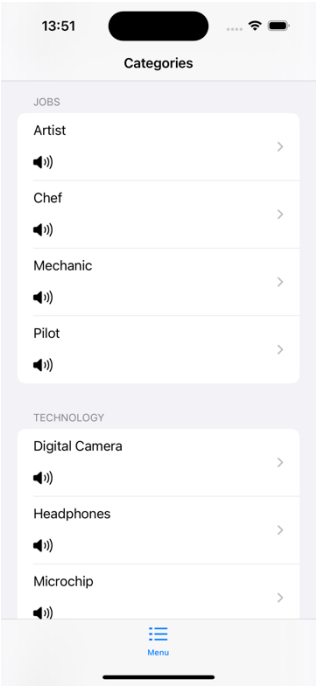
3.1.6.2. Dinamik Veri Güncellemeleri ve Kullanıcı Etkileşimi

WordListView kullanıcı arayüzü, ViewModel'deki veri değişikliklerine bağlı olarak dinamik olarak güncellenir. Bu, @ObservedObject dekoratörü ile sağlanır. Böylece, ViewModel'de kelime listesi güncellendiğinde, bu değişiklikler otomatik olarak kullanıcı arayüzünde yansıtılır. Kullanıcı uygulamada yeni kelimeler eklediğinde veya mevcut kelimeleri güncellediğinde, bu değişiklikler anında WordListView tarafından yakalanır ve liste güncellenir.

3.1.6.3. Kullanıcı Deneyimi ve Arayüz Tasarımı

WordListView'ın kullanıcı arayüzü, okunabilirlik ve kullanıcı dostu navigasyonu ön planda tutacak şekilde tasarlanmıştır. Liste yapısı, kelimelerin kolayca taranmasını ve her bir kelimenin hızlıca incelenmesini sağlar. Bu sayede, kullanıcılar kendi kelime dağarcıklarını etkili bir şekilde gözden geçirebilir ve öğrenme süreçlerini daha iyi yönetebilir.

3.1.7. CategoryListView.swift (kaynak kod EK 7'de görülebilir)



Şekil 3.1.1: Uygulama Ana Ekranı

"CategoryListView.swift" dosyası, RepetiCards uygulamasında kategorilere göre düzenlenmiş kelimeleri görüntüleyen bir arayüz sunar. Bu dosya, kelimelerin daha sistematik ve düzenli bir şekilde incelenmesini sağlayarak kullanıcıların öğrenme sürecini destekler. Kategorilere ayrılmış kelimeler, kullanıcıların belirli konulara odaklanmasını ve öğrenme materyallerini daha etkili bir şekilde yönetmesini kolaylaştırır.

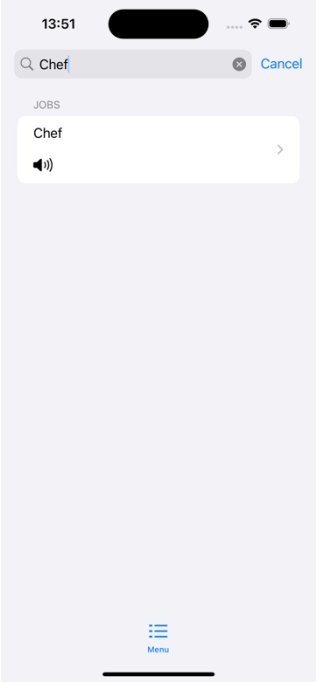
3.1.7.1. Kategorilere Göre Listeleme ve Arayüz Yapısı

Dosya içindeki CategoryListView struct'ı, NavigationView ve List kullanarak, FlashcardViewModel'den alınan kategori ve kelime verilerini görüntüler. Her kategori, bir Section içinde gruplandırılır ve bu kategorinin altındaki kelimeler listelenir. Bu yapı, kullanıcılara her kategorideki kelimeleri kolayca gözden geçirme ve inceleme fırsatı sunar.

3.1.7.2. Dinamik Veri Güncellemeleri ve Kullanıcı Etkileşimi

Bu görünüm, ViewModel'deki kategorilere göre düzenlenmiş kelimeleri dinamik olarak gösterir. @ObservedObject veya @State dekoratörleri kullanılarak, ViewModel'de gerçekleşen değişiklikler anında yansıtılır. Kullanıcılar uygulamada yeni kelimeler ekledikçe veya mevcut kelimeleri güncelledikçe, bu değişiklikler CategoryListView tarafından yakalanır ve kategori listesi güncellenir.

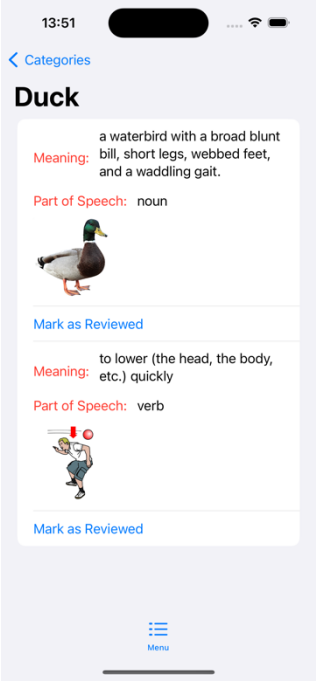
3.1.7.3. Arama Fonksiyonelliği ve Kullanıcı Deneyimi



Şekil 3.1.2: Arama Ekranı

Dosyada, kullanıcıların kategoriler arasında arama yapmalarını sağlayan bir arama çubuğu bulunur. Bu, @State ile yönetilen bir arama metni değişkeni kullanılarak sağlanır. Kullanıcıların arama terimlerine göre kelimeleri filtrelemeleri, istedikleri kelimelere daha hızlı ulaşmalarını sağlar ve öğrenme deneyimini daha interaktif ve kişiselleştirilmiş hale getirir.

3.1.8. MeaningListView.swift" (kaynak kod EK 8'de görülebilir)



Şekil 3.1.3: Kelime Detay Ekranı

"MeaningListView.swift" dosyası, RepetiCards uygulamasında seçilen bir kelimenin anlamlarını listelemek ve detaylarını göstermek için kullanılır. Bu dosya, kullanıcılara her bir kelimenin çeşitli anlamlarını, örneklerini ve ilgili görsel/işitsel materyalleri sunarak öğrenme sürecini zenginleştirir. Bu görünüm, kelimelerin derinlemesine incelenmesini sağlar ve kullanıcının kelime dağarcığını daha etkin bir şekilde geliştirmesine yardımcı olur.

3.1.8.1. Anlam Listesi ve Navigasyon Yapısı

Dosya içinde tanımlanan MeaningListView struct'ı, bir kelimenin tüm anlamlarını bir List içinde gösterir. Bu liste, her anlam için detaylı bilgileri ve örnek kullanımları içerir. Kullanıcılar, bu liste aracılığıyla her anlamı ayrı ayrı inceleyebilir ve dil bilgisi, kullanım örnekleri ve görsel/işitsel referanslar hakkında bilgi edinebilirler.

3.1.8.2. Dinamik Veri Güncellemeleri ve İşlevsellik

Bu görünüm, seçilen kelimenin anlamlarını dinamik bir şekilde günceller ve FlashcardViewModel aracılığıyla verileri alır. Kullanıcılar tarafından yapılan herhangi bir güncelleme veya eklemeler, ViewModel'deki değişikliklerle senkronize edilir ve anlam listesi buna göre güncellenir.

3.1.8.3. Kullanıcı Etkileşimleri ve Görsel/İşitsel Unsurlar

Her anlam için, kullanıcıların kelimenin telaffuzunu dinlemelerini sağlayan işitsel düğmeler ve kelimeyle ilişkili görseller bulunur. Bu özellikler, öğrenme deneyimini zenginleştirir ve kelimelerin daha iyi hatırlanmasını sağlar. Kullanıcılar, her anlamın yanındaki butonlar aracılığıyla kelimelerin telaffuzunu dinleyebilir ve görsel ögelerle kelimenin anlamını pekiştirebilir.

3.1.9. ReviewWordsView.swift (kaynak kod EK 9’da görülebilir)

"ReviewWordsView.swift" dosyası, RepetiCards uygulamasında kullanıcıların gözden geçirmeleri gereken kelimeleri listelemek için tasarlanmıştır. Bu dosya, aralıklı tekrarlama tekniğine dayalı öğrenme metodolojisinin bir parçası olarak, kullanıcılara belirlenen

zamanlarda kelime tekrarlarını hatırlatır ve gözden geçirmelerini sağlar. Bu sayede, kullanıcıların kelime hatırlama ve pekiştirme süreçleri desteklenir, böylece öğrenilen kelimeler daha kalıcı hale getirilir.

3.1.9.1. Gözden Geçirme Listesi ve Navigasyon Yapısı

Dosyada tanımlanan ReviewWordsView struct'ı, bir List kullanarak FlashcardViewModel'den alınan, gözden geçirilmesi gereken kelimeleri görüntüler. Bu liste, kullanıcıların her bir kelime ve onun anlamlarını detaylıca incelemesine ve gözden geçirmesine olanak tanır. Kullanıcıların her kelimenin anlamını ve kullanımını gözden geçirmesi, öğrenme sürecinde önemli bir adımdır.

3.1.9.2. Dinamik Veri Güncellemeleri ve Kullanıcı Etkileşimi

ReviewWordsView arayüzü, ViewModel'deki gözden geçirilmesi gereken kelimeler listesini dinamik olarak günceller. Kullanıcıların uygulamada yaptığı etkileşimler, ViewModel'de değişikliklerle senkronize olur ve bu değişiklikler liste üzerinde yansıtılır. Kullanıcılar, gözden geçirdikleri kelimeleri işaretledikçe, bu bilgiler ViewModel'de güncellenir ve kullanıcının öğrenme ilerlemesi takip edilir.

3.1.9.3. Kullanıcı Deneyimi ve Öğrenme Süreci

Dosyanın kullanıcı arayüzü, kullanıcıların gözden geçirmeleri gereken kelimeleri kolayca görebilmeleri ve erişebilmeleri için tasarlanmıştır. Her kelime, kullanıcıya kelimenin anlamını ve kullanımını hatırlatır ve gözden geçirme sürecini kolaylaştırır. Bu yaklaşım, kullanıcıların öğrenme sürecini daha etkili ve düzenli hale getirir.

3.1.10. "NotificationManager.swift" (kaynak kod EK 10'da görülebilir)

Dosyada tanımlanan NotificationManager sınıfı, kullanıcıdan bildirim gönderme izni istemek için bir metod içerir. Bu izin talebi, kullanıcının uygulama ayarlarında yapılandırılır ve kullanıcının bildirimleri alma tercihine saygı gösterir. Kullanıcı bildirimlere izin verirse, uygulama belirlenen zamanlarda öğrenme hatırlatmaları gönderebilir.

3.1.10.1. Bildirim Zamanlaması ve Gönderimi

NotificationManager sınıfı, belirlenen tarih ve saatlerde özel bildirimler göndermek için özelleştirilmiş tetikleyiciler ve içerikler kullanır. scheduleNotification metodu, bir kelimenin gözden geçirme zamanı geldiğinde kullanıcıya bildirim göndermek için kullanılır. Bu metod, kullanıcıya özel bir bildirim içeriği oluşturur ve bu içeriği belirlenen tarihte göndermek üzere zamanlar.

3.1.10.2. Kullanıcı Etkileşimi ve Bildirimler

NotificationManager ayrıca kullanıcının bildirimlere nasıl tepki verdiğini yönetir. Bildirimlere tıklanıldığında, kullanıcı uygulamaya yönlendirilir ve ilgili kelime veya öğrenme materyali gösterilir. Bu, kullanıcı etkileşimini artırır ve öğrenme sürecine aktif katılımı teşvik eder.

3.1.10.3. Güvenlik ve Performans

Bu dosya, uygulamanın bildirim sistemini güvenli ve verimli bir şekilde yönetmek için gelişmiş programlama teknikleri ve iOS bildirim framework'ünü kullanır. Bildirimlerin doğru şekilde yönetilmesi ve zamanlanması, kullanıcı deneyiminin kalitesini artırır ve uygulamanın genel performansına katkıda bulunur.

3.1.11. AudioPlayerService.swift (kaynak kod EK 11'de görülebilir)

"AudioPlayerService.swift" dosyası, RepetiCards uygulamasında ses dosyalarını yönetmek ve oynatmak için kullanılan bir hizmet sınıfıdır. Bu dosya, öğrenme sürecini desteklemek ve kullanıcı deneyimini zenginleştirmek için kelime telaffuzları ve dil öğreniminde önemli olan diğer sesli materyalleri oynatma işlevini üstlenir. Sesli öğrenme, özellikle dil öğreniminde, kelime hatırlama ve telaffuz becerilerinin geliştirilmesi için kritik bir rol oynar.

3.1.11.1. Ses Oynatma Yöntemleri ve AVFoundation Kullanımı

Dosya içinde tanımlanan AudioPlayerService sınıfı, iOS'un güçlü AVFoundation framework'ünü kullanarak ses dosyalarını yönetir ve oynatır. Bu sınıf, ses dosyalarını yüklemek, oynatmak ve ses oynatıcıyı yönetmek için metodlar içerir. playSound metodu, belirli bir ses dosyasını (örneğin, bir kelimenin telaffuzunu) oynatmak için kullanılır ve bu dosyanın uygulama içerisindeki kaynağını belirler.

3.1.11.2. Kullanıcı Etkileşimi ve Sesli Materyaller

AudioPlayerService sınıfı, kullanıcıların uygulama içindeki kelime kartları üzerindeki ses simgelerine tıkladığında kelime telaffuzlarını oynatır. Bu, öğrenme sürecini interaktif hale getirir ve kullanıcılara kelimelerin doğru telaffuzlarını duyma imkanı sunar. Ses oynatma işlevi, kullanıcıların dil öğreniminde telaffuz ve dinleme becerilerini geliştirmesine yardımcı olur.

3.1.11.3. Performans ve Güvenilirlik

Bu dosya, ses oynatma işlemlerini verimli ve hatasız bir şekilde gerçekleştirmek için AVAudioPlayer sınıfının yeteneklerinden faydalanır. Ses dosyalarının yüklenmesi ve oynatılması sırasında oluşabilecek hatalar düşünülerek dikkatli bir hata yönetimi yapısı kurulmuştur. Bu yaklaşım, uygulamanın sesli öğelerinin güvenilir ve kesintisiz bir şekilde çalışmasını sağlar.

4. Sonuçlar

"RepetiCards" uygulamasının geliştirilmesi ve kullanıma sunulması, İngilizce kelime öğrenimi alanında önemli bir ilerleme olarak değerlendirilebilir. Uygulama, aralıklı tekrarlama tekniği ve kategorilere dayalı öğrenme yaklaşımını başarıyla entegre etmiştir. Bu yöntemlerin kullanımı, kullanıcıların kelime hatırlama oranlarında bir artış sağlayacağı ve öğrenme süreçlerini daha verimli hale getireceği tahmin edilmektedir.. Ayrıca, sesli materyaller ve görsel öğelerin entegrasyonu, dil öğrenimini daha etkileşimli ve eğlenceli bir deneyime dönüştürmüştür.

4.1. Kullanıcı Geri Bildirimleri ve Katılımı

Uygulamayı test eden kullanıcıların geri bildirimleri çoğunlukla olumludur. Uygulamanın kullanıcı dostu arayüzü, özelleştirilmiş öğrenme planları ve etkili öğrenme araçları test eden kullanıcılar tarafından olumlu karşılanmıştır.. Kullanıcılar, özellikle sesli telaffuz özelliklerini ve kelime anlamlarının görselleştirilmesini yararlı bulmuşlardır. Ayrıca, kullanıcılar tarafından yapılan öneriler, uygulamanın sürekli geliştirilmesi için değerli bir kaynak olmuştur.

4.2. Başarı Ölçütleri ve Öğrenme İlerlemesi

Uygulamanın etkinliği, kullanıcıların kelime hatırlama başarı oranları ve öğrenme ilerlemesi üzerinden değerlendirilebilir. Aralıklı tekrarlama yönteminin kullanılmasının uzun vadeli kelime hatırlama başarısını önemli ölçüde artırması beklenmektedir.

4.3. Sonuç ve Gelecek Planları

Sonuç olarak, "RepetiCards" uygulaması, İngilizce kelime öğrenimi alanında etkili bir araç olabilir. Uygulamanın kullanıcı dostu tasarımı, etkileşimli öğrenme araçları ve bilimsel temelli öğrenme teknikleri, dil öğrenimini desteklemekte ve kullanıcıların öğrenme deneyimlerini zenginleştirmekte önemli rol oynamaktadır. Gelecekte, uygulama daha fazla dil seçeneği, gelişmiş özelleştirme ayarları ve kullanıcı etkileşimlerine dayalı özelliklerle zenginleştirilecek, böylece öğrenme deneyimi daha da iyileştirilecektir.

Kaynaklar

Clark, R. C., & Mayer, R. E. (2016). e-Learning and the Science of Instruction: Proven Guidelines for Consumers and Designers of Multimedia Learning. Wiley.

Crompton, H. (2013). Mobile learning: New approach, new theory. In Z. L. Berge & L. Y. Muilenburg (Eds.), Handbook of mobile learning (pp. 47-57). Routledge.

Ebbinghaus, H. (1885). Memory: A Contribution to Experimental Psychology. Dover Publications.

Kapler, I. V., Weston, T., & Wiseheart, M. (2015). Spacing in a simulated undergraduate classroom: Long-term benefits for factual and higher-level learning. Learning and Instruction, 36, 38-45.

Kukulska-Hulme, A. (2010). Mobile learning as a catalyst for change. Open Learning: The Journal of Open, Distance and e-Learning, 25(3), 181-185.

Mayer, R. E. (2001). Multimedia Learning. Cambridge University Press.

Nielsen, J., & Norman, D. (2000). Interaction Design. Wiley.

Wozniak, P. A. (1990). Theoretical aspects of spaced repetition in learning. Good Language Learner.

Ekler

EK-1 RepetiCardsApp.swift

```
import SwiftUI
import UserNotifications


@main
struct RepetiCardsApp: App {
   let persistenceController = CoreDataStack.shared
   let viewModel = FlashcardViewModel()


   init() {
        NotificationManager.shared.requestAuthorization()
     }


   var body: some Scene {
      WindowGroup {
         TabView {
```

```
            ContentView(viewModel: viewModel)
                .tabItem {
                    Label("Menu", systemImage: "list.dash")
                }

            AddWordView(viewModel: viewModel)
                .tabItem {
                    Label("Order", systemImage: "square.and.pencil")
                }
        }
    }
}
```

EK-2 CoreDataStack.swift

```
import Foundation
import CoreData


class CoreDataStack {


    static let shared = CoreDataStack()


    lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "RepetiCards")
        container.loadPersistentStores(completionHandler: { (storeDescription, error) in
            if let error = error as NSError? {
                fatalError("Unresolved error \(error), \(error.userInfo)")
            }
        })
        return container
    }()
```

```
    var context: NSManagedObjectContext {
        return persistentContainer.viewContext
    }

    func saveContext () {
        let context = persistentContainer.viewContext
        if context.hasChanges {
            do {
                try context.save()
            } catch {
                let nserror = error as NSError
                fatalError("Unresolved error \(nserror), \(nserror.userInfo)")
            }
        }
    }
}
```

EK-3 ContentView.swift

```
import SwiftUI


struct ContentView: View {
    var viewModel = FlashcardViewModel()


    var body: some View {
        NavigationView {
            CategoryListView(viewModel: viewModel)
            //WordListView(viewModel: viewModel)
        }
    }
}
```

EK-4 FlashcardRepository.swift

```
import CoreData
import Foundation


class FlashcardRepository {


    let context: NSManagedObjectContext


    init(context: NSManagedObjectContext = CoreDataStack.shared.context) {
        self.context = context
    }
    //MARK: - CATEGORY RELATED
    func createCategory(name: String) -> CardCategory {
        let category = CardCategory(context: context)
        category.name = name
        saveContext()
        return category
    }
```

```
    func assignCategory(to meaning: Meaning, category: CardCategory) {
        meaning.cardCategory = category
        saveChanges()
    }
//

    func getAllWords() -> [Word] {
        let wordFetchRequest: NSFetchRequest<Word> = Word.fetchRequest()
        do {
            return try context.fetch(wordFetchRequest)
        } catch {
            print("Error fetching words: \(error)")
            return []
        }
    }

    func getMeanings(for word: Word) -> [Meaning] {
        // Assuming 'meanings' is the name of the relationship from Word to Meaning
```

```
        let meaningsSet = word.meanings as? Set<Meaning> ?? []
        return Array(meaningsSet)
    }

    func addMeaning(to word: Word, definition: String, partOfSpeech: String, imageURL:
String?,category:CardCategory?) {
        let newMeaning = Meaning(context: context)
        newMeaning.id = UUID()
        newMeaning.definition = definition
        newMeaning.partOfSpeech = partOfSpeech
        newMeaning.imageURL = imageURL
        newMeaning.reviewDate = Date() // Set initial review date to current date
        newMeaning.word = word
        newMeaning.cardCategory = category

        saveChanges()
    }

    func updateReviewDate(for meaning: Meaning, with newDate: Date) {
        meaning.reviewDate = newDate
```

```
    saveChanges()
}

func updateReviewForMeaning(meaning: Meaning, with nextReviewDate: Date) {
    meaning.reviewCount += 1
    meaning.reviewDate = nextReviewDate

    // Reset the isReviewed flag for the next cycle
    meaning.isReviewed = false

    saveChanges()
    print(meaning.isReviewed)
}

private func saveContext() {
    do {
        try context.save()
        print("Saved")
    } catch {
```

```
            print("Error saving context: \(error)")
        }
    }

    func saveChanges() {
        saveContext()
    }
}
```

EK-5 FlashcardViewModel.swift

```
import Foundation

import Combine

class FlashcardViewModel: ObservableObject {

    @Published var words: [Word] = []
    @Published var wordsForReview: [Word] = []
    @Published var wordsByCategory: [String: [Word]] = [:]

    var repository: FlashcardRepository
    private var cancellables: Set<AnyCancellable> = []

    init(repository: FlashcardRepository = FlashcardRepository()) {
        self.repository = repository
        loadWords()
    }
```

```
func loadWords() {
    words = repository.getAllWords()
   // loadDummyData()
}

func loadDummyData() {
    if words.count == 0 {
       print("CoreData is Empty.USING DUMMY DATA")
       self.addDummyWords()
   } else {
       print("USING CORE DATA")
   }
}

func organizeWordsByCategory() {
        var categoryMap: [String: Set<Word>] = [:]

        for word in words {
```

```
            for meaning in word.meaningsArray {
                guard let categoryName = meaning.cardCategory?.name else { continue }
                categoryMap[categoryName, default: Set<Word>()].insert(word)
            }
        }

        // Convert to [String: [Word]] for easier use in SwiftUI
    wordsByCategory = categoryMap.mapValues { Array($0).sorted { $0.word ?? "" < $1.word ?? "" } }
    }



  func scheduleNextReview(for meaning: Meaning) {
    // Example logic to determine the next review date based on the repetition count
    let nextReviewDate: Date
    print("Review Count \(meaning.reviewCount)")
    switch meaning.reviewCount {
    case 0:
        nextReviewDate = Calendar.current.date(byAdding: .day, value: 1, to: Date())!
    case 1:
```

```
            nextReviewDate = Calendar.current.date(byAdding: .day, value: 1, to: Date())!
        case 2:
            nextReviewDate = Calendar.current.date(byAdding: .day, value: 7, to: Date())!
        case 3:
            nextReviewDate = Calendar.current.date(byAdding: .day, value: 16, to: Date())!
        default:
            nextReviewDate = Calendar.current.date(byAdding: .day, value: 35, to: Date())!
        }

        NotificationManager.shared.scheduleNotification(for: meaning, on: nextReviewDate)
        repository.updateReviewForMeaning(meaning: meaning, with: nextReviewDate)
    }

    func showPendingNotifications() {
        NotificationManager.shared.showPendingNotifications()
    }

   /* func loadWordsForReview() {
        //let today = Date()
```

```
        wordsForReview = words.filter { word in
            word.meaningsArray.contains { meaning in
                guard let reviewDate = meaning.reviewDate else { return false }
                return Calendar.current.isDateInToday(reviewDate)
            }
        }
    }*/
    func loadWordsForReview() {
        print("Running loadWordsForReview")
        let today = Date()
        wordsForReview = words.map { word in
            let updatedMeanings = word.meaningsArray.map { meaning in
                print("Review Date \(meaning.reviewDate?.formatted(date: .long, time: .shortened))")
                print("Current Date: \(Date().formatted(date: .long, time: .shortened))")
                if let reviewDate = meaning.reviewDate, reviewDate < today {
                    meaning.isReviewed = false
                    saveChanges()
                }
                return meaning
```

```
            }
            word.meanings = NSSet(array: updatedMeanings)
            return word
        }
        // Trigger a UI update
        self.wordsForReview = wordsForReview
    }



    // Method to mark a flashcard's meaning as reviewed and calculate the next review date
    func markMeaningAsReviewed(meaning: Meaning) {
        // Logic to calculate the next review date goes here
        // For example, add one day to the current reviewDate
       //  if let reviewDate = meaning.reviewDate {
      //       meaning.reviewDate = Calendar.current.date(byAdding: .minute, value: 1, to: reviewDate)
       // }
        meaning.isReviewed = true
        repository.saveChanges()
        loadWordsForReview()
```

```
    }


    func triggerUpdate() {
        // This is a simple way to trigger a view update
        self.wordsForReview = Array(self.wordsForReview)
      }


    func addMeaning(to word: Word, definition: String, partOfSpeech: String, imageURL: String?,category:CardCategory) {
      repository.addMeaning(to: word, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageURL,category: category)
      loadWords()
    }


    /*  func addWord(word: String, definition: String, partOfSpeech: String, imageURL: String) {
     // Call the private repository method to add the word and its meaning
     let newWord = Word(context: repository.context)
     newWord.word = word
     repository.addMeaning(to: newWord, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageURL)
```

```
    loadWords()
    }*/


   func addWord(word: String, definition: String, partOfSpeech: String, imageURL: String?) {
        // Check if the word already exists
        if let existingWord = words.first(where: { $0.word == word }) {
            // Add the new meaning to the existing word
            repository.addMeaning(to: existingWord, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageURL,category: nil)
        } else {
            // Create a new word and add the meaning
            let newWord = Word(context: repository.context)
            newWord.word = word
            repository.addMeaning(to: newWord, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageURL,category: nil)
        }
        loadWords() // Refresh the words list
    }
```

```
    func addWord(word: String, definition: String, partOfSpeech: String, imageName: String?,category:CardCategory?) {
        // Check if the word already exists
        if let existingWord = words.first(where: { $0.word == word }) {
            // Add the new meaning to the existing word
            repository.addMeaning(to: existingWord, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageName,category: category)
        } else {
            // Create a new word and add the meaning
            let newWord = Word(context: repository.context)
            newWord.word = word
            repository.addMeaning(to: newWord, definition: definition, partOfSpeech: partOfSpeech, imageURL: imageName, category: category)
        }
        loadWords() // Refresh the words list
    }


    func addDummyWords() {
     //   let duckImageUrl = Bundle.main.url(forResource: "duck", withExtension: "jpeg")
    //            let duckVerbImageUrl = Bundle.main.url(forResource: "duckVerb", withExtension: "png")
```

```
//    let routerImageUrl = Bundle.main.url(forResource: "router", withExtension: "jpeg")


    let animalCategory = repository.createCategory(name: "Animals")

    let technology = repository.createCategory(name: "Technology")

    let verbs = repository.createCategory(name: "Verbs")

    let jobs = repository.createCategory(name: "Jobs")


    addWord(word: "Duck", definition: "to lower (the head, the body, etc.) quickly", partOfSpeech: "verb", imageName: "duckVerb", category: verbs)

    addWord(word: "Duck", definition: "a waterbird with a broad blunt bill, short legs, webbed feet, and a waddling gait.", partOfSpeech: "noun", imageName: "duck",category: animalCategory)

    addWord(word: "Router", definition: "a device that forwards data packets to the appropriate parts of a computer network.", partOfSpeech: "noun", imageName: "router",category: technology)

    addWord(word: "Pilot", definition: "a person who operates the flying controls of an aircraft.", partOfSpeech: "noun", imageName: "pilot",category: jobs)

    addWord(word: "Pilot", definition: "be the pilot of (an aircraft or ship).", partOfSpeech: "verb", imageName: "pilotVerb",category: verbs)

    addWord(word: "Digital Camera", definition: "a type of camera that records images that can be looked at on a computer", partOfSpeech: "noun", imageName: "digitalCamera",category: technology)
```

addWord(word: "Microchip", definition: "a tiny wafer of semiconducting material used to make an integrated circuit", partOfSpeech: "noun", imageName: "microchip",category: technology)

addWord(word: "Microchip", definition: "implant a microchip under the skin of (a domestic animal) as a means of identification.", partOfSpeech: "verb", imageName: "microchipVerb",category: technology)

addWord(word: "Rabbit", definition: "a gregarious burrowing plant-eating mammal, with long ears, long hind legs, and a short tail.", partOfSpeech: "noun", imageName: "rabbit",category: animalCategory)

addWord(word: "Parrot", definition: "a bird, often vividly coloured, with a short downcurved hooked bill, grasping feet, and a raucous voice, found especially in the tropics and feeding on fruits and seeds.", partOfSpeech: "noun", imageName: "parrot",category: animalCategory)

addWord(word: "Artist", definition: "a person who creates paintings or drawings as a profession or hobby.", partOfSpeech: "noun", imageName: "artist",category: jobs)

addWord(word: "Mechanic", definition: "a skilled worker who repairs and maintains vehicle engines and other machinery.", partOfSpeech: "noun", imageName: "mechanic",category: jobs)

addWord(word: "Fish", definition: "a limbless cold-blooded vertebrate animal with gills and fins living wholly in water.", partOfSpeech: "noun", imageName: "fishNoun",category: animalCategory)

addWord(word: "Fish", definition: "catch or try to catch fish, typically by using a net or hook and line.", partOfSpeech: "verb", imageName: "fishVerb",category: verbs)

addWord(word: "Headphones", definition: "a pair of earphones joined by a band placed over the head, for listening to audio signals such as music or speech.", partOfSpeech: "noun", imageName: "headphones",category: technology)

```
        addWord(word: "Chef", definition: "a professional cook, typically the chief cook in a restaurant or hotel.", partOfSpeech: "noun", imageName: "chef",category: jobs)

        //



    }

    func updateReviewDate(for meaning: Meaning, with newDate: Date) {
        repository.updateReviewDate(for: meaning, with: newDate)
    }

    func saveChanges() {
        repository.saveChanges()
    }
}
```

EK-6 WordListView.swift

```
import Foundation
import SwiftUI
struct WordListView: View {
    @ObservedObject var viewModel: FlashcardViewModel
    var body: some View {
        List(viewModel.words, id: \.self) { word in
            NavigationLink(destination: MeaningListView(word: word,viewModel: viewModel)) {
                Text(word.word ?? "")
            }
        }
        .navigationTitle("Words")
        .onAppear {
            //viewModel.loadWords()
            viewModel.loadWordsForReview()
        }
    }
}
```

EK-7 CategoryListView.swift

```
import SwiftUI

struct CategoryListView: View {
    @ObservedObject var viewModel: FlashcardViewModel
    @State private var searchText = ""


    var filteredWordsByCategory: [String: [Word]] {
        if searchText.isEmpty {
            return viewModel.wordsByCategory
        } else {
            var filtered: [String: [Word]] = [:]
            for (category, words) in viewModel.wordsByCategory {
                let filteredWords = words.filter { $0.word?.lowercased().contains(searchText.lowercased()) ?? false }
                if !filteredWords.isEmpty {
                    filtered[category] = filteredWords
                }
            }
            return filtered
```

```
        }
    }

    var body: some View {
        NavigationView {
            List {
                ForEach(filteredWordsByCategory.keys.sorted(), id: \.self) { categoryName in
                    Section(header: Text(categoryName)) {
                        ForEach(filteredWordsByCategory[categoryName] ?? [], id: \.self) { word in
                            NavigationLink(destination: MeaningListView(word: word, viewModel: viewModel)) {
                                VStack(alignment: .leading) {
                                    Text(word.word ?? "Unknown Word")
                                        .padding(.bottom,10)

                                    Button(action: {
                                        if let audioFilename = word.word {
                                            AudioPlayerService.shared.playSound(named: audioFilename)
                                        }
```

```
                        }) {
                            Image(systemName: "speaker.3.fill")
                                .resizable()
                                .aspectRatio(contentMode: .fit)
                                .frame(width: 24, height: 24)
                        }
                        .simultaneousGesture(TapGesture().onEnded {
                            print("Should play audio")
                            if let audioFilename = word.word {
                                AudioPlayerService.shared.playSound(named:
audioFilename)
                            }
                        })
                    }
                }
            }
        }
    }
    .navigationTitle("Categories")
```

```
        }
        .searchable(text: $searchText)
        .onAppear {
            viewModel.loadWordsForReview()
            viewModel.organizeWordsByCategory()
        }
    }
}
```

EK-8 MeaningListView.swift

```
import SwiftUI
struct MeaningListView: View {
    var word: Word
    @ObservedObject var viewModel: FlashcardViewModel



    var body: some View {
        List(word.meaningsArray, id: \.self) { meaning in
            VStack(alignment: .leading) {
                HStack {
                    Text("Meaning: ")
                        .foregroundStyle(.red)
                    Text(meaning.definition ?? "No definition")
                        .padding(.bottom,10)
                }

                HStack {
```

```
                Text("Part of Speech: ")
                    .foregroundStyle(.red)
                Text("\(meaning.partOfSpeech ?? "N/A")")
            }

            // Assuming imageURL is an optional string, you want to verify it's not nil or empty
            if let imageName = meaning.imageURL, !imageName.isEmpty
               {
               Image(imageName)
                  .resizable()
                  .frame(width: 100, height: 100)
            }
          /*    if let imageUrlString = meaning.imageURL, !imageUrlString.isEmpty, let imageUrl = URL(string: imageUrlString) {

                AsyncImage(url: imageUrl) { image in
                   image.resizable()
                } placeholder: {
                   ProgressView()
```

```
            }
            .frame(width: 100, height: 100)
        }*/
    }

    Button(meaning.isReviewed ? "Reviewed" : "Mark as Reviewed") {
        print("Reviewed button tapped for \(meaning.definition ?? "unknown")")
        if !meaning.isReviewed  {
            viewModel.scheduleNextReview(for: meaning)
            viewModel.showPendingNotifications()
            viewModel.markMeaningAsReviewed(meaning: meaning)
        } else {
            viewModel.showPendingNotifications()
        }
        viewModel.triggerUpdate()
    }

}
.navigationTitle(word.word ?? "Word")
```

```
    }
}

extension Word {
    var meaningsArray: [Meaning] {
        // Convert the NSSet to a sorted array of Meaning objects
        (meanings as? Set<Meaning> ?? []).sorted(by: { $0.definition ?? "" < $1.definition ?? "" })
    }
}
```

EK-9 ReviewWordsView.swift

```
import SwiftUI

struct ReviewWordsView: View {
    @ObservedObject var viewModel: FlashcardViewModel


    var body: some View {
        List(viewModel.wordsForReview, id: \.self) { word in
            VStack(alignment: .leading) {
                Text(word.word ?? "Unknown Word")
                ForEach(word.meaningsArray, id: \.self) { meaning in
                    HStack {
                        Text(meaning.definition ?? "No definition")
                        Spacer()
                        Button("Deneme") {
                            viewModel.markMeaningAsReviewed(meaning: meaning)
                        }
                    }
                }
            }
```

```
        }
        .onAppear {
            viewModel.loadWordsForReview()
        }
    }
}
```

EK-10 NotificationManager.swift

```
import Foundation
import UserNotifications


class NotificationManager: NSObject, UNUserNotificationCenterDelegate {
    static let shared = NotificationManager()


    private override init() {
        super.init()
        UNUserNotificationCenter.current().delegate = self
    }


    func requestAuthorization() {
        UNUserNotificationCenter.current().requestAuthorization(options: [.alert, .badge, .sound]) { granted, error in
            if let error = error {
                print("Notification Permission Error: \(error)")
            }
            // Handle the granted response if needed
        }
```

```
    }

    func scheduleNotification(for meaning: Meaning, on date: Date) {
        let content = UNMutableNotificationContent()
        guard let word = meaning.word?.word else {return}
        content.title = "\(word)"
        content.body = "\(meaning.definition ?? "")"
        content.sound = UNNotificationSound.default

        let triggerDate = Calendar.current.dateComponents([.year, .month, .day, .hour, .minute], from: date)
        let trigger = UNCalendarNotificationTrigger(dateMatching: triggerDate, repeats: false)

        let request = UNNotificationRequest(identifier: UUID().uuidString, content: content, trigger: trigger)
        UNUserNotificationCenter.current().add(request) { (error) in
            if let error = error {
                // Handle any errors
                print("Error scheduling notification: \(error)")
            }
```

```
        }
    }

    func showPendingNotifications() {
        UNUserNotificationCenter.current().getPendingNotificationRequests { pendingRequests in
            print("Active pending request count: \(pendingRequests.count)")
            for request in pendingRequests {
                print(request.content.title)
                print(request.content.body)
                print(request.trigger)
                print("------------------------------")
            }
        }
    }


    // UNUserNotificationCenterDelegate methods
    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                didReceive response: UNNotificationResponse,
```

```
                    withCompletionHandler completionHandler: @escaping () -> Void)
{
        // Handle the notification response
        completionHandler()
    }

    func userNotificationCenter(_ center: UNUserNotificationCenter,
                    willPresent notification: UNNotification,
                    withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void) {
        // Handle the notification presentation
        completionHandler([.banner, .sound])
    }
}
```

EK-11 AudioPlayerService.swift

```
import Foundation
import AVFoundation


class AudioPlayerService {
    static let shared = AudioPlayerService()
    var player: AVAudioPlayer?


    func playSound(named soundName: String) {
        guard let url = Bundle.main.url(forResource: soundName, withExtension: "m4a")
else { return }


        do {
            player = try AVAudioPlayer(contentsOf: url)
            player?.play()
        } catch {
            print("Could not load file: \(error)")
        }
    }
}
```