24/12/2025

Proje Gerçekleştirim Raporu

Mobil Cihazlarda Yapay Zekâ Destekli Egzersiz Formu Doğrulama ve Raporlama Uygulamasının Geliştirilmesi

Mehmet Salih AYDIN

222132022

İÇİNDEKİLER TABLOSU

1. GİRİŞ

Bu bölümde uygulamanın hangi amaçlarla geliştirildiği, bu süreçte hangi teknolojilerden yararlanıldığı ve geliştirme ortamının nasıl yapılandırıldığı açıklanmaktadır. Gerçekleştirim aşaması, analiz ve tasarım evrelerinde belirlenen gereksinimlerin somut bir yazılım ürününe dönüştürüldüğü aşama olup, uygulamanın teknik altyapısının ve kullanıcıya sunulan işlevselliğin fiilen hayata geçirildiği süreci kapsamaktadır. Bu aşamada yalnızca uygulamanın çalışır hale getirilmesi değil, aynı zamanda **sürdürülebilir**, **genişletilebilir** ve **bakımı kolay** bir yazılım mimarisinin oluşturulması hedeflenmiştir. Uygulamanın farklı mobil platformlarda tutarlı bir kullanıcı deneyimi sunması da gerçekleştirim sürecinin temel hedefleri arasında yer almaktadır.

1.1. GERÇEKLEŞTİRİM AŞAMASININ AMACI

Gerçekleştirim aşamasının temel amacı, sistemin analiz ve tasarım safhalarında tanımlanan gereksinimlerin, seçilen mimari yapı doğrultusunda **kodlanarak çalışan bir mobil uygulamaya dönüştürülmesidir**. Bu süreçte yalnızca işlevsel gereksinimlerin karşılanması değil, aynı zamanda uygulamanın uzun vadede geliştirilebilirliğini destekleyen bir kod yapısının oluşturulması ön planda tutulmuştur.

Uygulamanın **durum yönetimi**, **yerel veri saklama**, **çoklu dil desteği** ve **platform bağımsız çalışma** gibi temel özellikleri, gerçekleştirim sürecinde bütüncül bir bakış açısıyla ele alınmıştır. Ayrıca kullanıcı etkileşimlerine hızlı tepki verebilen, akıcı ve anlaşılır bir kullanıcı arayüzü sunan bir yapı oluşturulması amaçlanmıştır. Bu doğrultuda, iş mantığı ile kullanıcı arayüzünün birbirinden ayrıldığı modern yazılım geliştirme yaklaşımları benimsenmiştir.

1.2. KULLANILAN TEKNOLOJİLER VE ARAÇLAR

Gerçekleştirim sürecinde, güncel mobil uygulama geliştirme yaklaşımlarına uygun ve yaygın olarak kullanılan teknolojiler tercih edilmiştir. Uygulama, **Flutter framework'ü** kullanılarak geliştirilmiş olup tek bir kod tabanı üzerinden hem **iOS** hem de **Android** platformlarını destekleyecek şekilde tasarlanmıştır. Flutter'ın sunduğu yüksek performanslı kullanıcı arayüzü altyapısı, uygulamanın görsel tutarlılığını ve akıcılığını artırmıştır.

Uygulama genelinde **BLoC (Business Logic Component)** mimarisi kullanılarak durum yönetimi sağlanmıştır. Bu sayede iş mantığı ile kullanıcı arayüzü birbirinden ayrılmış, uygulamanın test edilebilirliği ve bakım kolaylığı artırılmıştır. Yerel veri yönetimi için **Hive** veritabanı tercih edilmiş olup, uygulama verilerinin hızlı ve güvenli bir şekilde cihaz üzerinde saklanması sağlanmıştır. Ayrıca uygulamanın çoklu dil desteğine sahip olması amacıyla **Easy Localization** kütüphanesi kullanılarak kullanıcıya farklı dil seçenekleri sunulmuştur.

1.3. GELİŞTİRME ORTAMI VE PLATFORM SEÇİMİ

Uygulamanın geliştirilmesi sürecinde **çapraz platform** yaklaşımı benimsenmiş ve bu doğrultuda Flutter framework'ü tercih edilmiştir. Bu seçim, aynı uygulamanın hem iOS hem de Android platformları için ayrı ayrı geliştirilmesi ihtiyacını ortadan kaldırarak geliştirme sürecini daha verimli ve yönetilebilir hale getirmiştir. Böylece platformlar arasında tutarlı bir kullanıcı deneyimi sunulması mümkün olmuştur.

Geliştirme süreci, modern entegre geliştirme ortamları kullanılarak yürütülmüş; kodlama, test ve hata ayıklama işlemleri bu ortamlar üzerinden gerçekleştirilmiştir. Uygulama, sanal cihazlar ve fiziksel mobil cihazlar üzerinde test edilerek farklı platform ve senaryolarda doğru şekilde çalıştığı doğrulanmıştır. Mobil platformların sunduğu kamera ve sensör gibi donanımsal özelliklerin uygulamanın temel işlevselliğinde önemli bir rol oynaması, platform seçiminin mobil odaklı yapılmasında belirleyici olmuştur.

2. SİSTEM MİMARİSİ VE TASARIM KARARLARI

Bu bölümde uygulamanın yazılım mimarisi ele alınmakta, geliştirme sürecinde alınan temel tasarım kararları ve bu kararların arkasındaki gerekçeler açıklanmaktadır. Mobil uygulamanın yalnızca çalışır olması değil; **ölçeklenebilir**, **bakımı kolay** ve **uzun vadede geliştirilebilir** bir yapıya sahip olması hedeflendiğinden, mimari tasarım sürecine özel bir önem verilmiştir.

Uygulamanın karmaşık iş mantığı içermesi, gerçek zamanlı veri işleme gereksinimi ve farklı bileşenlerin birbiriyle etkileşim halinde olması; mimari yapının belirli prensiplere dayalı olarak kurgulanmasını zorunlu kılmıştır. Bu doğrultuda, modern yazılım geliştirme yaklaşımlarından yararlanılarak net sınırlarla ayrılmış bir mimari yapı tercih edilmiştir.

2.1. GENEL MİMARİ YAKLAŞIM

Uygulamanın genel mimarisi, **ayrık sorumluluklar (separation of concerns)** ilkesine dayalı olarak tasarlanmıştır. İş mantığı, veri yönetimi ve kullanıcı arayüzü katmanlarının birbirinden bağımsız olması hedeflenmiş; her katmanın yalnızca kendi sorumluluk alanına odaklanması sağlanmıştır.

Bu yaklaşım sayesinde, uygulamanın belirli bir bölümünde yapılacak değişikliklerin diğer bileşenleri minimum düzeyde etkilemesi amaçlanmıştır. Ayrıca test süreçlerinin daha verimli yürütülmesi ve ileride eklenecek yeni özelliklerin mevcut yapıyı bozmadan entegre edilebilmesi hedeflenmiştir. Bu bağlamda, uygulama mimarisi **Clean Architecture** prensipleri temel alınarak oluşturulmuştur.

2.1.1. CLEAN ARCHİTECTURE YAKLAŞIMI

Uygulama geliştirilirken **Clean Architecture** yaklaşımı benimsenmiş ve sistem bu mimari prensiplere uygun şekilde yapılandırılmıştır. Clean Architecture, yazılımın merkezine iş kurallarını koyarak dış bağımlılıkların bu çekirdeğe bağımlı olmasını öngören bir yaklaşımdır. Bu sayede uygulamanın iş mantığı; kullanıcı arayüzü, veri kaynakları veya platform detaylarından bağımsız hale getirilmiştir.

Bu yaklaşımın tercih edilmesindeki temel neden, uygulamanın **test edilebilirliğini artırmak**, **teknoloji bağımlılığını azaltmak** ve **uzun vadeli bakım maliyetlerini düşürmek** olmuştur. Uygulamanın ilerleyen aşamalarda yeni veri kaynaklarıyla çalışabilmesi veya farklı bir arayüz yapısına geçebilmesi, Clean Architecture sayesinde mümkün hale gelmiştir.

2.1.2. KATMANLI MİMARİ YAPISI

Clean Architecture yaklaşımı doğrultusunda uygulama, mantıksal olarak birbirinden ayrılmış katmanlar halinde tasarlanmıştır. Bu katmanlar genel olarak **sunum (presentation)**, **alan (domain)** ve **veri (data)** katmanlarından oluşmaktadır. Her katman, yalnızca kendi sorumluluk alanına ait işlemleri gerçekleştirmekte ve diğer katmanlarla sınırlı bir etkileşim içinde bulunmaktadır.

Sunum katmanı, kullanıcı arayüzü ve durum yönetimi ile ilgilenirken; alan katmanı uygulamanın temel iş kurallarını ve mantığını içermektedir. Veri katmanı ise yerel veri saklama ve veri erişim işlemlerinden sorumludur. Bu yapı sayesinde, uygulamanın işleyişi daha anlaşılır hale gelmiş ve kod organizasyonu belirli bir düzen içerisinde tutulmuştur.

2.1.3. KATMANLAR ARASI BAĞIMLILIK KURALLARI

Uygulama mimarisinde katmanlar arası bağımlılıklar **tek yönlü** olacak şekilde tasarlanmıştır. Üst katmanlar, alt katmanlara doğrudan bağımlı olmamakta; bunun yerine **arayüzler (interfaces)** üzerinden iletişim kurmaktadır. Bu durum, özellikle alan katmanının dış dünyadan tamamen bağımsız kalmasını sağlamaktadır.

Bağımlılıkların bu şekilde sınırlandırılması, uygulamanın farklı bileşenlerinin birbirinden kopuk bir şekilde test edilebilmesine olanak tanımaktadır. Aynı zamanda veri kaynağının veya kullanıcı arayüzünün değiştirilmesi gibi durumlarda, iş mantığının bu değişikliklerden etkilenmemesi sağlanmıştır. Bu tasarım kararı, uygulamanın **esnek**, **bakımı kolay** ve **genişletilebilir** bir yapıya sahip olmasında önemli bir rol oynamıştır.

2.2. PROJE KATMANLARININ YAPILANDIRILMASI

Uygulamanın yazılım mimarisi, Clean Architecture yaklaşımına uygun olarak **net sorumluluk sınırlarına sahip katmanlar** şeklinde yapılandırılmıştır. Bu yapı sayesinde kod okunabilirliği artırılmış, geliştirme süreci daha kontrollü hale getirilmiş ve uzun vadeli bakım kolaylaştırılmıştır. Her katman, yalnızca kendi görev alanına odaklanmakta ve diğer katmanlarla doğrudan bağımlılık oluşturmadan iletişim kurmaktadır.

Katmanların bu şekilde ayrılması, hem ekip içi geliştirme sürecini kolaylaştırmış hem de uygulamanın test edilebilirliğini ve genişletilebilirliğini önemli ölçüde artırmıştır.

2.2.1. PRESENTATİON KATMANI

Presentation katmanı, uygulamanın **kullanıcı arayüzü** ve **kullanıcı etkileşimleri** ile ilgili tüm bileşenleri içermektedir. Bu katmanda Flutter widget'ları, ekranlar ve durum yönetimini sağlayan **BLoC/Cubit** yapıları yer almaktadır. Kullanıcıdan gelen etkileşimler bu katmanda karşılanmakta ve gerekli iş mantığı çağrıları ilgili katmanlara iletilmektedir.

Bu katmanın temel sorumluluğu, kullanıcıya doğru ve güncel bilgiyi sunmak olup; iş kurallarına dair herhangi bir doğrudan mantık barındırmamaktadır. Presentation katmanı, domain katmanına yalnızca **arayüzler (use case'ler)** üzerinden erişmekte ve böylece iş mantığından soyutlanmış bir yapı sunmaktadır. Bu yaklaşım, kullanıcı arayüzünde yapılacak değişikliklerin sistemin diğer bölümlerini etkilemeden gerçekleştirilebilmesini sağlamaktadır.

2.2.2. DOMAİN KATMANI

Domain katmanı, uygulamanın **merkezi iş mantığını** ve temel kurallarını barındıran en kritik katmandır. Bu katmanda; varlıklar (entities), iş kurallarını temsil eden use case'ler ve soyut repository arayüzleri yer almaktadır. Domain katmanı, hiçbir şekilde Flutter, veri tabanı veya harici servis detaylarına bağımlı değildir.

Bu bağımsız yapı sayesinde uygulamanın iş mantığı, platformdan ve teknolojiden tamamen ayrıştırılmıştır. Böylece domain katmanı, farklı veri kaynaklarıyla veya farklı sunum katmanlarıyla çalışabilecek esnekliğe sahiptir. Bu yaklaşım, uygulamanın **test edilebilirliğini** ve **uzun vadeli sürdürülebilirliğini** önemli ölçüde artırmaktadır.

2.2.3. DATA KATMANI

Data katmanı, uygulamanın **veri yönetimi** ile ilgili tüm işlemlerinden sorumludur. Yerel veri saklama işlemleri, veri modelleri ve repository implementasyonları bu katmanda yer almaktadır. Uygulamada verilerin kalıcı olarak saklanması için **Hive** kullanılmış ve bu yapı domain katmanında tanımlanan repository arayüzlerini somutlaştıracak şekilde uygulanmıştır.

Bu katman, domain katmanına veri sağlamakla yükümlüdür ancak domain katmanının iç işleyişi hakkında bilgi sahibi değildir. Veri kaynaklarının değiştirilmesi veya farklı bir veri saklama çözümüne geçilmesi durumunda, yalnızca data katmanında düzenleme yapılması yeterli olmaktadır. Bu da mimarinin **esnekliğini** ve **bakım kolaylığını** artırmaktadır.

2.2.4. CORE VE ORTAK BİLEŞENLER

Core katmanı, uygulamanın tüm katmanları tarafından ortak olarak kullanılan **yardımcı servisler**, **sabitler**, **tema yapılandırmaları**, **bağımlılık enjeksiyonu** ve **yardımcı araçları** içermektedir. Bu katmanda yer alan bileşenler, belirli bir iş alanına özgü olmayıp uygulamanın genelinde kullanılmaktadır.

Bağımlılık enjeksiyonu bu katmanda merkezi olarak yönetilmiş, servislerin ve repository'lerin uygulama genelinde tutarlı bir şekilde erişilmesi sağlanmıştır. Ayrıca tema, renk paletleri ve ortak UI bileşenleri bu katmanda toplanarak görsel tutarlılık korunmuştur. Core katmanı, uygulamanın genel mimarisini destekleyen temel yapı taşı niteliğindedir.

2.3. DURUM YÖNETİMİ (STATE MANAGEMENT)

Uygulamada kullanıcı etkileşimleri, kamera akışı, egzersiz analizleri ve veri güncellemeleri gibi birçok dinamik süreç eş zamanlı olarak yönetilmektedir. Bu karmaşık yapı içerisinde, arayüz ile iş mantığı arasındaki veri akışının kontrollü ve tutarlı bir şekilde sağlanabilmesi için merkezi bir **durum yönetimi yaklaşımı** benimsenmiştir. Bu amaçla uygulamada **BLoC (Business Logic Component)** mimarisi tercih edilmiştir.

2.3.1. BLOC MİMARİSİ SEÇİM GEREKÇESİ

BLoC mimarisi, uygulamanın iş mantığını kullanıcı arayüzünden tamamen ayıran bir yapı sunduğu için tercih edilmiştir. Bu yaklaşım sayesinde UI bileşenleri yalnızca mevcut durumu dinlemekte ve kullanıcıdan gelen aksiyonları event olarak iletmektedir. İş mantığına dair tüm kararlar ise BLoC katmanında alınmaktadır.

BLoC yapısı; **ölçeklenebilirlik**, **test edilebilirlik** ve **okunabilirlik** açısından güçlü avantajlar sağlamaktadır. Özellikle gerçek zamanlı kamera verisi ile çalışan ve sürekli state güncellemesi gerektiren Pose Vision uygulamasında, bu mimari sayesinde karmaşık durum geçişleri kontrol altında tutulmuştur. Ayrıca Flutter ekosistemiyle olan yüksek uyumluluğu, tercih edilmesinde etkili olmuştur.

2.3.2. EVENT – STATE – BLOC İLİŞKİSİ

BLoC mimarisinde uygulama akışı, **Event → BLoC → State** zinciri üzerinden ilerlemektedir. Kullanıcı tarafından gerçekleştirilen her etkileşim veya sistem tarafından tetiklenen her işlem bir event olarak tanımlanmakta ve ilgili BLoC'a iletilmektedir. BLoC, bu event'i işleyerek gerekli iş mantığını çalıştırmakta ve sonucunda yeni bir state üretmektedir.

Üretilen state, kullanıcı arayüzüne tek yönlü bir veri akışıyla aktarılmaktadır. Bu yapı sayesinde uygulama içerisindeki tüm durum değişimleri izlenebilir, tahmin edilebilir ve yönetilebilir hale gelmiştir. Event ve state kavramlarının açık bir şekilde ayrılması, kodun anlaşılabilirliğini artırmış ve hata ayıklama süreçlerini kolaylaştırmıştır.

2.3.3. UI İLE STATE SENKRONİZASYONU

Kullanıcı arayüzü ile uygulama durumu arasındaki senkronizasyon, Flutter'ın sunduğu **BlocBuilder** ve **BlocConsumer** yapıları aracılığıyla sağlanmıştır. UI bileşenleri, yalnızca kendilerini ilgilendiren state değişikliklerini dinleyerek yeniden çizilmekte, gereksiz rebuild işlemlerinin önüne geçilmektedir.

Bu yaklaşım sayesinde uygulama performansı korunmuş, özellikle kamera ekranı gibi yüksek frekanslı güncellemelerin olduğu alanlarda stabil ve akıcı bir kullanıcı deneyimi elde edilmiştir. UI katmanı, state'e doğrudan müdahale etmemekte; yalnızca state değişimlerine tepki veren pasif bir yapı olarak konumlandırılmıştır.

2.3.4. UYGULAMA GENELİNDE STATE YÖNETİMİ

Uygulama genelinde farklı sorumluluklara sahip birden fazla BLoC ve Cubit yapısı kullanılmıştır. Egzersiz süreci, tema ayarları ve geçmiş verilerin yönetimi gibi alanlar birbirinden ayrılarak, her biri kendi state yapısı içerisinde ele alınmıştır. Bu ayrım, modülerliği artırmış ve uygulamanın farklı bölümlerinin birbirinden bağımsız şekilde geliştirilmesine olanak tanımıştır.

State yönetiminin bu şekilde merkezi ve tutarlı bir yapıda ele alınması, uygulamanın **bakım kolaylığını**, **genişletilebilirliğini** ve **kararlılığını** önemli ölçüde artırmıştır. Ayrıca bu yapı, Clean Architecture yaklaşımıyla uyumlu bir şekilde, iş mantığı ile kullanıcı arayüzü arasındaki sınırların net olarak korunmasını sağlamıştır.

3. UYGULAMA GERÇEKLEŞTİRİMİ

Bu bölümde Pose Vision uygulamasının geliştirme sürecinde alınan teknik kararlar, uygulamanın nasıl hayata geçirildiği ve farklı platformlarda nasıl çalışır hale getirildiği ele alınmaktadır. Uygulama, hem kullanıcı deneyimi hem de teknik sürdürülebilirlik açısından modern mobil geliştirme yaklaşımları doğrultusunda gerçekleştirilmiştir.

3.1. ÇOKLU PLATFORM DESTEĞİ (İOS & ANDROİD)

Pose Vision uygulaması, **tek bir kod tabanı** üzerinden hem iOS hem de Android platformlarını destekleyecek şekilde geliştirilmiştir. Bu yaklaşım sayesinde platformlar arası tutarlılık sağlanmış, geliştirme ve bakım maliyetleri önemli ölçüde azaltılmıştır. Uygulamanın tüm temel işlevleri her iki platformda da aynı davranışı gösterecek biçimde tasarlanmıştır.

Flutter framework'ünün sunduğu platform bağımsız yapı sayesinde, uygulama farklı işletim sistemlerinde doğal (native) bir deneyim sunabilmektedir. Kamera erişimi, sensör kullanımı ve performans gereksinimleri gibi platforma özgü konular ise kontrollü bir şekilde ele alınmıştır.

3.1.1. FLUTTER İLE TEK KOD TABANI YAKLAŞIMI

Uygulama geliştirme sürecinde **Flutter** tercih edilerek, Dart dili kullanımıyla tek bir kod tabanı üzerinden iki farklı mobil platform hedeflenmiştir. Bu sayede iş mantığı, kullanıcı arayüzü ve veri yönetimi gibi bileşenler için ayrı ayrı platform geliştirme ihtiyacı ortadan kaldırılmıştır.

Tek kod tabanı yaklaşımı; geliştirme sürecinin hızlanmasını, hata oranının azalmasını ve yeni özelliklerin her iki platforma eş zamanlı olarak eklenmesini mümkün kılmıştır. Ayrıca kod tekrarının önüne geçilerek daha temiz ve yönetilebilir bir proje yapısı elde edilmiştir.

3.1.2. PLATFORM BAĞIMSIZ UI BİLEŞENLERİ

Uygulamanın kullanıcı arayüzü, Flutter'ın sunduğu **platformdan bağımsız widget yapısı** kullanılarak oluşturulmuştur. Tüm ekranlar, bileşenler ve etkileşimler ortak UI bileşenleri üzerinden geliştirilmiş, platforma özgü farklılıklar minimum seviyede tutulmuştur.

Bu yaklaşım sayesinde iOS ve Android kullanıcıları arasında tutarlı bir görsel dil ve etkileşim deneyimi sağlanmıştır. Aynı zamanda tema yönetimi, renk paleti ve tipografi gibi görsel unsurlar merkezi bir yapıdan kontrol edilerek uygulamanın genel tasarım bütünlüğü korunmuştur.

3.1.3. CİHAZ VE DONANIM UYUMLULUĞU

Pose Vision uygulaması, farklı ekran boyutlarına ve donanım kapasitelerine sahip mobil cihazlarla uyumlu olacak şekilde tasarlanmıştır. Responsive tasarım prensipleri kullanılarak, UI bileşenlerinin çeşitli çözünürlüklerde doğru şekilde ölçeklenmesi sağlanmıştır.

Kamera kullanımı ve poz algılama gibi donanım yoğun işlemler, cihaz performansını olumsuz etkilemeyecek şekilde optimize edilmiştir. Android ve iOS platformlarında kamera yönelimi, sensör farkları ve görüntü formatları dikkate alınarak gerekli uyarlamalar yapılmış, böylece uygulamanın her iki platformda da stabil ve güvenilir şekilde çalışması sağlanmıştır.

3.2. KULLANICI ARAYÜZÜ (UI) VE KULLANICI DENEYİMİ (UX)

Pose Vision uygulamasında kullanıcı arayüzü ve kullanıcı deneyimi tasarımı, uygulamanın teknik yeteneklerini kullanıcıya sade, anlaşılır ve motive edici bir biçimde sunmayı hedefleyecek şekilde ele alınmıştır. Özellikle kamera tabanlı gerçek zamanlı egzersiz analizi yapan bir uygulama olması nedeniyle, arayüzün dikkat dağıtmayan, hızlı algılanabilir ve geri bildirim odaklı olması öncelikli tasarım kriterleri arasında yer almıştır.

Uygulama genelinde tutarlı bir görsel dil benimsenmiş, modern mobil tasarım yaklaşımları doğrultusunda hem estetik hem de işlevsel bir kullanıcı deneyimi oluşturulmuştur.

3.2.1. UI TASARIM PRENSİPLERİ

Uygulamanın arayüz tasarımı oluşturulurken **sadeliği**, **okunabilirliği** ve **odaklanmayı artıran** bir yapı benimsenmiştir. Kullanıcıların egzersiz sırasında ekrana kısa sürelerle bakacağı göz önünde bulundurularak, kritik bilgiler büyük puntolarla ve yüksek kontrastla sunulmuştur.

Renk kullanımı bilinçli şekilde sınırlandırılmış, başarı ve hata durumları için anlamlı renkler tercih edilmiştir. Örneğin doğru tekrarlar ve başarı durumları yeşil tonlarla, hatalı durumlar ise kırmızı tonlarla vurgulanarak kullanıcıya sezgisel geri bildirim sağlanmıştır. Cam efekti (glassmorphism) gibi modern tasarım öğeleri, uygulamanın görsel kalitesini artırırken arayüzün profesyonel bir görünüm kazanmasına katkı sağlamıştır.

3.2.2. RESPONSİVE VE ADAPTİVE TASARIM

Pose Vision, farklı ekran boyutlarına ve çözünürlüklere sahip cihazlarda sorunsuz çalışacak şekilde **responsive tasarım** prensipleriyle geliştirilmiştir. UI bileşenleri, sabit ölçüler yerine ekran boyutuna göre ölçeklenen yapılar kullanılarak tasarlanmıştır.

Ayrıca iOS ve Android platformları arasındaki küçük etkileşim ve görünüm farkları göz önünde bulundurularak adaptif davranışlar sergileyen bileşenler tercih edilmiştir. Bu sayede uygulama, her iki platformda da kullanıcıların alışık olduğu doğal bir deneyim sunabilmektedir.

3.2.3. ANİMASYONLAR VE GÖRSEL GERİ BİLDİRİMLER

Uygulamada kullanıcıyı motive etmek ve yapılan işlemlerin sonucunu anlık olarak göstermek amacıyla **animasyonlar ve görsel geri bildirimler** etkin biçimde kullanılmıştır. Bir tekrar tamamlandığında ekranda beliren doğru/yanlış simgeleri, yumuşak geçiş animasyonları ile desteklenmiştir.

Bu görsel geri bildirimler, kullanıcının yaptığı hareketin doğruluğunu anında fark etmesini sağlar ve egzersiz sırasında sözel anlatıma gerek kalmadan yönlendirme yapılmasına olanak tanır. Aynı zamanda animasyonlar abartıdan kaçınılarak, performansı olumsuz etkilemeyecek ve dikkat dağıtmayacak şekilde tasarlanmıştır.

3.2.4. ERİŞİLEBİLİRLİK VE KULLANILABİLİRLİK

Uygulamanın kullanılabilirliği artırmak amacıyla, **erişilebilirlik** ilkeleri göz önünde bulundurulmuştur. Büyük butonlar, yeterli dokunma alanları ve net ikonlar sayesinde farklı kullanıcı profilleri için kolay bir kullanım hedeflenmiştir.

Metinlerin okunabilirliği için yeterli kontrast sağlanmış, karmaşık navigasyon yapılarından kaçınılarak kullanıcıların minimum adımda hedeflerine ulaşabilmesi amaçlanmıştır. Ayrıca sesli geri bildirim (TTS) desteği sayesinde, ekrana sürekli bakma gereksinimi azaltılarak kullanıcı deneyimi daha güvenli ve konforlu hale getirilmiştir.

3.3. YERELLEŞTİRME (LOCALİZATİON) YAPISI

Pose Vision uygulaması, farklı dil ve bölgelere sahip kullanıcılar tarafından rahatlıkla kullanılabilmesi amacıyla **yerelleştirme (localization)** desteği ile geliştirilmiştir. Uygulamanın tüm metinsel içerikleri, dil bağımlı yapıdan ayrıştırılarak merkezi bir çeviri altyapısı üzerinden yönetilmiştir. Bu yaklaşım, hem kullanıcı deneyimini iyileştirmiş hem de uygulamanın ileride farklı dillere genişletilmesini kolaylaştırmıştır.

Yerelleştirme yapısı, sadece arayüz metinleriyle sınırlı tutulmamış; hata mesajları, bilgilendirme metinleri ve kullanıcıya gösterilen geri bildirimler de bu kapsama dahil edilmiştir.

3.3.1. ÇOKLU DİL DESTEĞİ ALTYAPISI

Uygulamada çoklu dil desteği sağlamak amacıyla Flutter ekosisteminde yaygın olarak kullanılan **Easy Localization** paketi tercih edilmiştir. Bu paket sayesinde tüm metinler, dil anahtarları (keys) üzerinden yönetilmiş ve uygulama içerisindeki sabit string kullanımı minimuma indirilmiştir.

Dil dosyaları JSON formatında yapılandırılmış olup her dil için ayrı çeviri dosyaları oluşturulmuştur. Bu yapı, çevirilerin merkezi bir noktadan kontrol edilmesini sağlarken, geliştirici tarafında da okunabilir ve sürdürülebilir bir kod yapısı sunmuştur.

3.3.2. DİL DEĞİŞİM MEKANİZMASI

Uygulama içerisinde kullanıcıların dili **çalışma zamanında (runtime)** değiştirebilmesine olanak tanıyan bir mekanizma tasarlanmıştır. Kullanıcı ayarlar ekranı üzerinden tercih ettiği dili seçtiğinde, uygulama yeniden başlatılmadan tüm arayüz metinleri seçilen dile anlık olarak güncellenmektedir.

Bu süreçte dil bilgisi uygulama genelinde state olarak ele alınmış ve dil değişikliği sonrası UI bileşenlerinin otomatik olarak yeniden çizilmesi sağlanmıştır. Böylece kullanıcı, kesintisiz ve akıcı bir deneyim yaşamaktadır.

3.3.3. UI VE VERİ KATMANINDA LOCALİZATİON KULLANIMI

Yerelleştirme yalnızca kullanıcı arayüzüyle sınırlı tutulmamış, **UI ve veri katmanı arasında tutarlı bir kullanım** sağlanmıştır. Özellikle hata mesajları, egzersiz geri bildirimleri ve uyarı metinleri, domain ve presentation katmanlarında doğrudan string yerine localization anahtarları üzerinden yönetilmiştir.

Bu yaklaşım sayesinde uygulama genelinde dil tutarsızlıkları engellenmiş, farklı ekranlarda aynı metnin farklı biçimlerde gösterilmesinin önüne geçilmiştir. Ayrıca ileride yeni bir dil eklenmesi gerektiğinde, uygulama mantığında herhangi bir değişiklik yapmadan yalnızca ilgili çeviri dosyalarının güncellenmesi yeterli olmaktadır.

4. VERİ YÖNETİMİ VE KALICILIK

Pose Vision uygulamasında kullanıcıya ait egzersiz verilerinin güvenli, hızlı ve sürdürülebilir bir şekilde saklanabilmesi için **yerel veri kalıcılığı** temel bir tasarım gereksinimi olarak ele alınmıştır. Uygulama, çevrimdışı senaryolarda da tam işlevsellik sunabilmeli ve geçmiş antrenman verilerine kesintisiz erişim sağlamalıdır. Bu doğrultuda, veri yönetimi katmanı performans, ölçeklenebilirlik ve bakım kolaylığı kriterleri gözetilerek tasarlanmıştır.

4.1. YEREL VERİ SAKLAMA ALTYAPISI

Uygulamanın veri saklama altyapısı, mobil cihazların donanım kısıtları ve performans beklentileri göz önünde bulundurularak yerel bir çözüm üzerine inşa edilmiştir. Egzersiz oturumları, tekrar kayıtları ve kullanıcı ayarları gibi veriler cihaz üzerinde saklanarak hem hızlı erişim hem de veri gizliliği sağlanmıştır.

Yerel veri saklama yaklaşımı, uygulamanın ağ bağlantısına bağımlı olmadan çalışabilmesini mümkün kılmış ve kullanıcı deneyimini önemli ölçüde iyileştirmiştir.

4.1.1. HİVE VERİTABANI SEÇİM GEREKÇESİ

Yerel veri saklama çözümü olarak **Hive** veritabanı tercih edilmiştir. Hive, Flutter ekosistemi ile yüksek uyumluluğa sahip, **NoSQL tabanlı** ve hafif bir veritabanı çözümüdür. Şema zorunluluğu olmadan çalışabilmesi, hızlı okuma-yazma işlemleri sunması ve düşük bellek tüketimi sağlaması bu tercihte belirleyici olmuştur.

Ayrıca Hive'ın **binary tabanlı depolama yapısı**, performans açısından avantaj sağlarken, adapter mekanizması sayesinde karmaşık veri modellerinin güvenli bir şekilde saklanmasına olanak tanımaktadır. Bu özellikler, Pose Vision uygulamasında anlık veri üretimi ve yoğun okuma işlemleri için uygun bir altyapı sunmuştur.

4.1.2. VERİ MODELLERİNİN YAPILANDIRILMASI

Uygulama içerisinde saklanan tüm veriler, açık ve tutarlı veri modelleri üzerinden yapılandırılmıştır. Egzersiz oturumları ve tekrar kayıtları gibi temel varlıklar için **ayrı veri modelleri** tanımlanmış ve bu modeller Hive adapter'ları aracılığıyla veritabanına entegre edilmiştir.

Her veri modeli, uygulamanın domain katmanındaki karşılıklarıyla uyumlu olacak şekilde tasarlanmış; böylece veri katmanı ile iş mantığı arasında net bir ayrım sağlanmıştır. Model yapılarında tip güvenliği ön planda tutulmuş ve enum gibi sabit yapılar da Hive uyumlu hale getirilmiştir.

Bu yaklaşım, hem veri tutarlılığını artırmış hem de ileride yapılabilecek genişletmeler için sağlam bir temel oluşturmuştur.

4.1.3. PERFORMANS VE VERİ ERİŞİM OPTİMİZASYONU

Veri erişiminde performansı artırmak amacıyla Hive'ın **Box** yapısı etkin bir şekilde kullanılmıştır. Okuma ve yazma işlemleri doğrudan anahtar (key) bazlı gerçekleştirildiği için veri erişim süreleri minimum seviyede tutulmuştur. Ayrıca, gereksiz veri kopyalarının önüne geçmek için yalnızca ihtiyaç duyulan veriler belleğe alınmıştır.

Egzersiz geçmişi gibi çoklu kayıt içeren veri setlerinde sıralama ve filtreleme işlemleri uygulama seviyesinde kontrol edilerek, veritabanı üzerindeki yük azaltılmıştır. Bu sayede hem bellek kullanımı optimize edilmiş hem de kullanıcı arayüzünde akıcı bir deneyim sağlanmıştır.

4.2. REPOSİTORY DESENİ

Pose Vision uygulamasında veri erişimi ve veri kaynaklarının yönetimi, **Repository deseni** kullanılarak soyutlanmıştır. Bu yaklaşım sayesinde uygulamanın iş mantığı, verinin nasıl ve nerede saklandığı bilgisinden tamamen bağımsız hale getirilmiştir. Repository deseni, Clean Architecture yaklaşımının temel taşlarından biri olarak veri katmanı ile domain katmanı arasında net ve kontrollü bir sınır oluşturmuştur.

Bu yapı, hem kodun okunabilirliğini artırmış hem de ileride farklı veri kaynaklarının eklenmesini kolaylaştırmıştır.

4.2.1. REPOSİTORY KATMANININ ROLÜ

Repository katmanı, uygulamadaki tüm veri işlemleri için **tek giriş noktası** olarak görev yapmaktadır. Egzersiz oturumlarının kaydedilmesi, tekrar kayıtlarının saklanması ve geçmiş verilerin getirilmesi gibi işlemler doğrudan repository üzerinden gerçekleştirilmiştir.

Bu katman, verinin Hive veritabanından mı, farklı bir yerel kaynaktan mı yoksa ileride eklenecek uzak bir servisten mi geldiğini üst katmanlardan gizlemektedir. Böylece presentation ve domain katmanları yalnızca ihtiyaç duydukları verilere odaklanmakta, veri erişim detaylarıyla ilgilenmemektedir.

4.2.2. DOMAİN VE DATA KATMANI ARASINDAKİ İLETİŞİM

Domain katmanı ile data katmanı arasındaki iletişim, **arayüz (interface)** temelli bir yapı üzerinden sağlanmıştır. Domain katmanında tanımlanan repository arayüzleri, data katmanında somut sınıflar tarafından uygulanmıştır.

Bu yaklaşım sayesinde domain katmanı, veri katmanına doğrudan bağımlı olmamış; yalnızca sözleşme (contract) üzerinden iletişim kurmuştur. Böylece bağımlılık yönü Clean Architecture prensiplerine uygun şekilde **dış katmanlardan iç katmanlara doğru** olacak biçimde düzenlenmiştir.

Ayrıca bu yapı, test edilebilirliği artırmış ve farklı veri kaynaklarının sisteme entegre edilmesini kolaylaştırmıştır.

4.2.3. VERİ OKUMA VE YAZMA AKIŞLARI

Veri okuma ve yazma işlemleri, belirli ve tutarlı bir akış üzerinden yönetilmiştir. Presentation katmanından gelen talepler, önce domain katmanındaki iş kurallarından geçmekte, ardından repository arayüzü aracılığıyla data katmanına iletilmektedir. Data katmanı ise bu talepleri Hive veritabanı üzerinde gerçekleştirmektedir.

Bu kontrollü akış sayesinde veri bütünlüğü korunmuş, tekrar eden kodlar azaltılmış ve uygulama genelinde tutarlı bir veri yönetim yapısı elde edilmiştir. Ayrıca hata yönetimi ve loglama işlemleri repository seviyesinde ele alınarak uygulamanın kararlılığı artırılmıştır.

5. EGZERSİZ VE ANALİZ MODÜLÜ GERÇEKLEŞTİRİMİ

Pose Vision uygulamasının temel işlevselliği, egzersizlerin doğru şekilde algılanması, analiz edilmesi ve kullanıcıya anlamlı geri bildirimler sunulması üzerine kuruludur. Bu nedenle egzersiz yönetimi ve analiz süreçleri, uygulamanın en kritik modüllerinden biri olarak ele alınmıştır. Egzersiz türleri, analiz mantığından bağımsız fakat onunla uyumlu olacak şekilde yapılandırılmıştır.

5.1. EGZERSİZ TÜRLERİNİN YÖNETİMİ

Uygulamada desteklenen egzersizler, genişletilebilir ve sürdürülebilir bir yapı hedeflenerek merkezi bir sistem üzerinden yönetilmiştir. Her egzersiz türü, hem kullanıcı arayüzünde hem de analiz motorunda tutarlı bir şekilde temsil edilmektedir. Bu yaklaşım sayesinde yeni bir egzersizin sisteme eklenmesi, mevcut yapıyı bozmadan mümkün hale getirilmiştir.

5.1.1. EGZERSİZ TİPLERİNİN TANIMLANMASI

Egzersiz türleri, uygulama genelinde tekil bir kaynak üzerinden tanımlanmıştır. Bu tanımlar, egzersizlerin kimliğini, analiz sürecindeki davranışını ve kullanıcı arayüzündeki temsilini belirleyen temel yapı taşlarıdır. Egzersiz tiplerinin **enum tabanlı** olarak modellenmesi, tip güvenliğini artırmış ve hatalı kullanım riskini azaltmıştır.

Bu yapı sayesinde uygulamanın farklı katmanlarında aynı egzersiz tanımı kullanılmakta, böylece veri tutarlılığı ve kod bütünlüğü sağlanmaktadır.

5.1.2. EGZERSİZ BAZLI KONFİGÜRASYONLAR

Her egzersiz için gerekli olan görsel, metinsel ve davranışsal özellikler, egzersiz bazlı konfigürasyonlar aracılığıyla yönetilmiştir. Bu konfigürasyonlar; egzersiz ikonları, renk gradyanları, lokalizasyon anahtarları ve açıklama metinleri gibi kullanıcı deneyimini doğrudan etkileyen unsurları içermektedir.

Bu merkezi konfigürasyon yapısı, **UI ve analiz katmanlarının birbirinden bağımsız çalışmasını** mümkün kılmıştır. Aynı zamanda kullanıcı arayüzünde tutarlı bir görünüm elde edilirken, analiz motoru egzersiz türüne özel kuralları kolaylıkla uygulayabilmektedir. Bu yaklaşım, bakım maliyetini düşürmüş ve uygulamanın ölçeklenebilirliğini artırmıştır.

5.2. GERÇEK ZAMANLI ANALİZ SÜRECİ

Uygulamanın temel katma değeri, kullanıcının egzersizleri gerçekleştirirken hareketlerinin **gerçek zamanlı olarak analiz edilmesi** ve doğru–yanlış geri bildirimlerin anında sunulmasıdır. Bu süreç; kamera erişimi, poz algılama, egzersize özgü analiz kuralları ve kullanıcıya sunulan görsel geri bildirimlerden oluşan bütünleşik bir yapı üzerine inşa edilmiştir.

5.2.1. KAMERA VE SENSÖR ENTEGRASYONU

Gerçek zamanlı analiz sürecinin ilk adımı, cihaz kamerasından alınan görüntülerin doğru ve kesintisiz şekilde işlenmesidir. Uygulama, mobil cihazın yerleşik kamerasını kullanarak kullanıcıdan anlık görüntü akışı elde etmektedir. Kamera erişimi, işletim sistemi seviyesinde izin mekanizmaları ile kontrol edilmekte ve kullanıcı onayı olmadan analiz süreci başlatılmamaktadır.

Kamera entegrasyonu sırasında performans ve gecikme faktörleri göz önünde bulundurulmuş, analiz sürecinin kullanıcı deneyimini olumsuz etkilememesi hedeflenmiştir. Bu doğrultuda görüntü verisi, yalnızca analiz için gerekli formatta ve gerçek zamanlı olarak işlenmekte, gereksiz veri saklama veya aktarımından kaçınılmaktadır.

5.2.2. HAREKET ALGILAMA VE ANALİZ AKIŞI

Kamera üzerinden elde edilen görüntüler, hareket algılama sürecine aktarılmadan önce uygun veri formatına dönüştürülmektedir. Bu aşamadan sonra poz algılama teknolojisi kullanılarak kullanıcının vücut eklem noktaları tespit edilir. Elde edilen bu noktalar, egzersiz türüne özel analiz kuralları ile değerlendirilir.

Analiz akışı; poz verisinin alınması, eklem ilişkilerinin hesaplanması ve egzersize özgü doğruluk kriterlerinin kontrol edilmesi adımlarından oluşmaktadır. Her tekrar, bağımsız bir analiz sürecinden geçirilerek doğru veya hatalı olarak sınıflandırılır. Bu yapı sayesinde egzersizlerin **nesnel ve tutarlı** bir şekilde değerlendirilmesi sağlanmıştır.

5.2.3. ANLIK GERİ BİLDİRİM MEKANİZMASI

Analiz sürecinin en önemli çıktısı, kullanıcıya sunulan **anlık geri bildirimlerdir**. Her tekrar tamamlandığında, analiz sonucu anında kullanıcı arayüzüne yansıtılmaktadır. Doğru tekrarlar olumlu görsel göstergelerle belirtilirken, hatalı tekrarlar için egzersize özgü hata mesajları sunulmaktadır.

Bu geri bildirim mekanizması, kullanıcının egzersiz sırasında kendi formunu anında düzeltmesine olanak tanımaktadır. Aynı zamanda sesli bildirim ve görsel vurgular ile desteklenerek, kullanıcının dikkatinin egzersizden kopmaması hedeflenmiştir. Böylece uygulama, yalnızca bir takip aracı değil, **aktif bir rehber** rolü üstlenmektedir.

6. UYGULAMA AYARLARI VE KİŞİSELLEŞTİRME

Mobil egzersiz uygulamalarında kullanıcı bağlılığını artıran en önemli unsurlardan biri, uygulamanın **kullanıcı tercihlerine uyum sağlayabilmesidir**. Bu doğrultuda geliştirilen sistem, görünümden egzersiz parametrelerine kadar birçok ayarın kullanıcı tarafından özelleştirilebilmesine olanak tanımaktadır. Ayarlar modülü, hem kullanıcı deneyimini iyileştirmeyi hem de uygulamanın farklı kullanım alışkanlıklarına adapte olmasını hedeflemektedir.

6.1. TEMA VE GÖRÜNÜM AYARLARI

Uygulama, kullanıcıların görsel tercihlerini dikkate alarak **tema ve görünüm ayarlarını** özelleştirebilecekleri bir yapı sunmaktadır. Açık tema, koyu tema ve sistem temasını takip eden modlar desteklenmektedir. Böylece kullanıcı, cihaz ayarlarıyla uyumlu bir deneyim elde edebilmekte ya da kendi tercihine göre sabit bir tema seçebilmektedir.

Tema yapısı yalnızca renk değişimi ile sınırlı tutulmamış; arka planlar, yüzey renkleri, vurgu tonları ve ikon kontrastları da tema değişimine uyumlu olacak şekilde tasarlanmıştır. Bu yaklaşım, hem görsel tutarlılığı korumakta hem de farklı ışık koşullarında okunabilirliği artırmaktadır.

6.2. VARSAYILAN EGZERSİZ PARAMETRELERİ

Uygulama içerisinde her egzersiz türü için **varsayılan tekrar sayıları** ve temel egzersiz parametreleri kullanıcı tarafından ayarlanabilmektedir. Kullanıcılar, squat, biceps curl ve lateral raise gibi egzersizler için hedef tekrar sayılarını önceden belirleyerek, egzersiz sürecini kendi seviyelerine uygun hale getirebilmektedir.

Bu yapı sayesinde uygulama, hem yeni başlayan kullanıcılar hem de ileri seviye kullanıcılar için esnek bir kullanım sunmaktadır. Varsayılan değerlerin kişiselleştirilebilir olması, kullanıcıların her egzersiz oturumunda tekrar tekrar ayar yapma ihtiyacını ortadan kaldırmakta ve kullanım akışını sadeleştirmektedir.

6.3. KULLANICI TERCİHLERİNİN SAKLANMASI

Kullanıcı tarafından yapılan tüm ayarlar ve tercihler, uygulama içerisinde **kalıcı olarak saklanmaktadır**. Tema seçimi, egzersiz hedefleri ve diğer kullanıcı tercihleri, yerel depolama altyapısı üzerinden güvenli bir şekilde muhafaza edilmektedir. Uygulama her başlatıldığında bu tercihler otomatik olarak yüklenerek kullanıcıya tutarlı bir deneyim sunulmaktadır.

Bu yaklaşım, kullanıcı deneyimini güçlendirirken aynı zamanda uygulamanın profesyonel ve olgun bir ürün hissi vermesini sağlamaktadır. Kullanıcı ayarlarının kalıcı olması, uygulamanın kişisel bir yardımcı gibi davranmasına katkıda bulunmaktadır.

7. GENEL DEĞERLENDİRME

Bu çalışmada, modern mobil uygulama geliştirme yaklaşımları temel alınarak **çok katmanlı, ölçeklenebilir ve kullanıcı odaklı** bir egzersiz takip uygulaması başarıyla gerçekleştirilmiştir. Uygulama; yazılım mimarisi, durum yönetimi, veri kalıcılığı, kullanıcı arayüzü ve kişiselleştirme gibi başlıca alanlarda güncel teknolojileri bir araya getirerek bütüncül bir sistem sunmaktadır. Gerçekleştirilen yapı, hem akademik gereksinimleri karşılamakta hem de gerçek dünyada kullanılabilir bir ürün niteliği taşımaktadır.

7.1. GERÇEKLEŞTİRİLEN ÖZELLİKLERİN ÖZETİ

Geliştirilen uygulama, **Flutter** framework'ü kullanılarak iOS ve Android platformlarında tek kod tabanı üzerinden çalışacak şekilde tasarlanmıştır. Kullanıcılar; farklı egzersiz türlerini seçebilmekte, hedef tekrar sayılarını belirleyebilmekte ve egzersiz sırasında gerçek zamanlı analiz desteği almaktadır. Egzersiz oturumları tamamlandığında, detaylı performans özetleri ve tekrar bazlı analizler kullanıcıya sunulmaktadır.

Uygulama içerisinde **çoklu dil desteği (localization)** sağlanmış, kullanıcılar uygulama dilini dinamik olarak değiştirebilmektedir. Tema ve görünüm ayarları sayesinde açık, koyu ve sistem teması desteklenmiş; renk vurguları ve görsel bileşenler bu ayarlara uyumlu hale getirilmiştir. Tüm kullanıcı tercihleri ve egzersiz verileri yerel veritabanı üzerinden kalıcı olarak saklanarak, uygulamanın her açılışında tutarlı bir deneyim sağlanmıştır.

7.2. MİMARİ VE TEKNOLOJİK KAZANIMLAR

Bu proje kapsamında **Clean Architecture yaklaşımı** benimsenmiş ve katmanlar arası bağımlılıkların net bir şekilde ayrıldığı sürdürülebilir bir yapı oluşturulmuştur. Presentation, Domain ve Data katmanlarının birbirinden ayrılması, kodun okunabilirliğini ve test edilebilirliğini önemli ölçüde artırmıştır. İş mantığının UI'dan bağımsız hale getirilmesi, gelecekte yapılabilecek geliştirmeler için sağlam bir zemin oluşturmuştur.

Durum yönetimi için **BLoC mimarisi** kullanılarak, UI ile iş mantığı arasındaki ilişki kontrollü ve öngörülebilir bir yapıya kavuşturulmuştur. Event–State temelli bu yaklaşım, karmaşık kullanıcı etkileşimlerinin dahi yönetilebilir olmasını sağlamıştır. Veri kalıcılığı tarafında **Hive** veritabanının tercih edilmesi, yüksek performanslı ve düşük gecikmeli bir yerel veri erişimi sunmuştur.

Sonuç olarak bu çalışma, modern mobil uygulama geliştirme prensiplerini uygulamalı olarak ele alan, mimari açıdan güçlü ve genişletilebilir bir sistem ortaya koymuştur. Elde edilen mimari ve teknolojik kazanımlar, benzer ölçekteki mobil projeler için referans niteliği taşımaktadır.

8. SONUÇ

Bu proje kapsamında, modern mobil uygulama geliştirme prensipleri temel alınarak **fonksiyonel, ölçeklenebilir ve sürdürülebilir** bir egzersiz takip uygulaması başarıyla hayata geçirilmiştir. Uygulama, mimari tasarımdan kullanıcı deneyimine kadar bütün aşamalarda planlı ve sistematik bir yaklaşımla geliştirilmiş; akademik gereksinimler ile gerçek dünya ihtiyaçları dengeli bir şekilde ele alınmıştır.

8.1. PROJENİN GENEL BAŞARIMI

Geliştirilen sistem, **Flutter** ile tek kod tabanı kullanılarak iOS ve Android platformlarında çalışabilir hale getirilmiş; bu sayede platform bağımsız bir çözüm elde edilmiştir. **Clean Architecture** yaklaşımı sayesinde katmanlar arası sorumluluklar net bir biçimde ayrılmış, kodun okunabilirliği ve bakım kolaylığı önemli ölçüde artırılmıştır.

Durum yönetiminde kullanılan **BLoC mimarisi**, uygulama genelinde öngörülebilir ve kontrollü bir state yönetimi sağlamış; kullanıcı etkileşimleri ile iş mantığı arasındaki ilişki açık ve yönetilebilir bir yapıya kavuşturulmuştur. Yerel veri kalıcılığı için tercih edilen **Hive veritabanı**, performanslı veri okuma ve yazma işlemleri sunarak kullanıcı verilerinin güvenli şekilde saklanmasını mümkün kılmıştır.

Ayrıca uygulamada **çoklu dil desteği**, tema ve görünüm ayarları, varsayılan egzersiz parametreleri ve kişiselleştirme seçenekleri gibi kullanıcı odaklı özellikler başarıyla entegre edilmiştir. Tüm bu bileşenler bir araya getirildiğinde, proje hem teknik hem de kullanıcı deneyimi açısından hedeflenen başarı düzeyine ulaşmıştır.

8.2. GELECEKTE YAPILABİLECEK GELİŞTİRMELER

Mevcut uygulama, temel işlevleri eksiksiz şekilde yerine getirmekte olup, gelecekte genişletilmeye açık bir mimariye sahiptir. İlerleyen çalışmalarda; egzersiz analizlerinin daha gelişmiş algoritmalarla desteklenmesi, farklı egzersiz türlerinin sisteme eklenmesi ve analiz doğruluğunun artırılması planlanabilir.

Bunun yanında, **bulut tabanlı senkronizasyon**, kullanıcı hesapları ve cihazlar arası veri paylaşımı gibi özellikler eklenerek uygulamanın kullanım alanı genişletilebilir. Gelişmiş istatistikler, grafiksel performans raporları ve uzun vadeli ilerleme takibi gibi özellikler, kullanıcı motivasyonunu artıracak ek geliştirmeler arasında değerlendirilebilir.

Son olarak, erişilebilirlik ve kullanıcı deneyimi tarafında yapılacak iyileştirmeler ile uygulama daha geniş bir kullanıcı kitlesine hitap edecek şekilde geliştirilebilir. Bu yönüyle proje, gelecekte yapılacak çalışmalar için sağlam bir temel sunmaktadır.