T.C.
KIRKLARELİ ÜNİVERSİTESİ
TEKNİK BİLİMLER MESLEK YÜKSEKOKULU
BİLGİSAYAR TEKNOLOJİLERİ BÖLÜMÜ
BİLGİSAYAR PROGRAMCILIĞI PROGRAMI

KARŞILAŞTIRMALI STATE MANAGEMENT ANALİZİ

KÜBRA CELEP
1247008010

MOBİL PROGRAMLAMA
NADİR SUBAŞI
KIRKLARELİ
08/2025

İÇİNDEKİLER

1-Giriş

Flutter uygulamalarında durum yönetimi (state management), ekranlarda gösterilen verilerin kontrol edilmesi, güncellenmesi ve uygulama boyunca tutarlı bir şekilde aktarılması için kritik bir öneme sahiptir. Uygulamalar büyüdükçe, verilerin nerede tutulacağı, nasıl güncelleneceği ve hangi widget'ların bu değişikliklere tepki vereceği daha karmaşık hâle gelir. Bu nedenle geliştiriciler, uygulamanın performanslı, düzenli ve sürdürülebilir olmasını sağlamak için farklı durum yönetimi yöntemleri kullanırlar.

Bu projede Flutter'da en çok kullanılan üç durum yönetimi yaklaşımı olan Provider, Riverpod ve BLoC ele alınmış. Her birinin mantığı, ekosistemi, avantajları, dezavantajları ve birbirleriyle karşılaştırmaları detaylı şekilde açıklanmıştır. Amaç, bu üç yöntemin güçlü ve zayıf yönlerini anlamanı sağlayarak, projelerde hangi yapıyı tercih edileceği konusunda doğru karar verilmesine yardımcı olmaktır.

2-Provider nedir?

Provider, uygulamadaki verileri tek bir merkezde toplayan bir depo gibi çalışır. Bu yapı sayesinde, uygulamanın en üst katmanında tanımlanan bir veriye, en alttaki küçük bir widget bile karmaşık bağlantılar kurmadan kolayca ulaşabilir. Böylece veri yönetimi hem daha düzenli hâle gelir hem de uygulama içinde veri aktarımı hızlı ve güvenli bir yapıda sağlanmış olur.

2.1 Provider'in Temel Mantığı

Provider'in temel mantığı, uygulamadaki durumu merkezi bir yapı içinde yönetmek ve bu duruma ihtiyaç duyan widget'lara context üzerinden güvenli ve hızlı bir şekilde erişim sağlamaktır.

Durum değiştiğinde yalnızca o durumu kullanan widget'ların yeniden çizilmesini sağlar. Bunu genellikle **ChangeNotifier** (iş mantığını kullanıcı arayüzünden yönetmenin ve soyutlamanın bir yoludur) sınıfı ile birlikte çalışarak gerçekleştirir. Bu yapı sayesinde durum yönetimi daha düzenli, hızlı ve takip edilebilir bir hale gelir.

2.2 Provider Ekosistemi

ChangeNotifier = Veriyi tutan ve ne zaman değiştiğini takip eden sınıftır. Değişiklik olduğunda **notifyListeners()** metodu ile bu veriye bağlı olan tüm widget'ları güncellemesi için bilgilendirir.

Provider = **ChangeNotifier** sınıfını widget ağacına yerleştirir ve altındaki widget'ların bu duruma erişmesini sağlar.

Consumer / Selector = Durum (state) değişimlerini dinleyen ve değişime göre widget'ı yeniden inşa eden yapılardır. **Consumer** tüm state'e abone olurken, **Selector** state'in sadece belirli bir bölümü değiştiğinde yeniden inşa edilmeyi sağlar.

2.3 Neden Provider tercih ediliyor?

-Diğer karmaşık state management çözümlerine göre daha az karmaşıklık ve daha az tekrar eden kod gerektirir.

-İş mantığını (**ChangeNotifier**) UI'dan ayırdığı için, uygulamanın çekirdek mantığı kolayca test edilebilir.

-Flutter'ın en çok kullanılan durum yönetimi paketlerinden biridir. Bu yüzden hem dokümantasyonu çok güçlüdür hem de internette sayısız örnek, tutorial ve açıklama bulmak mümkündür.

2.4 Avantajlar ve Dezavantajlar

Avantajlar

- Flutter'ın yerleşik yapısını basitleştirir. Çekirdek bir flutter paketidir.

- Basit durumlar için çok hızlıdır.

Dezavantajlar

-Runtime hataları riski yüksektir.

-Veriye erişim için widget ağacının konumuna bağımlıdır.

-Performans optimizasyonu tamamen geliştiricinin sorumluluğundadır.

3-Riverpod nedir?

Riverpod, Provider paketini yazan kişi tarafından geliştirilmiştir ve Provider'ın sahip olduğu tüm sorunları ve kısıtlamaları çözmek üzere tasarlanmıştır. Bunlardan bazıları ise durum, bağımlılık yönetimi ve test edilebilirliğin daha kolay hale gelmesidir.

Riverpod, Provider'ın aksine widget'ların üzerinde değil, Provider'lar üzerinde çalışır.

3.1 Riverpod'un Temel Mantığı

Uygulamadaki veriyi (State) uygulamanın herhangi bir yerinden, widget ağacına veya **BuildContext**'e bağlı kalmadan erişilebilir hâle getirmeye dayanır. (BuildContext bir bileşenin widget ağacındaki konumunu temsil eden soyut bir sınıftır) Riverpod sayesinde veri yönetimi, widget ağacından bağımsız olarak merkezi ve güvenli bir şekilde gerçekleştirilir. Böylece küçük widget'lardan büyük ekran yapısına kadar tüm uygulamada veriye hızlı ve kolay bir şekilde erişilir.

3.2 Riverpod Ekosistemi

Provider = Uygulamanın ayar sabitleri, API anahtarları, servis nesneleri (Repository, Service sınıfları) gibi sadece bir kez oluşturulup değişmeyecek değerler.

StateProvider = Bir int, bool veya String gibi, sadece tek bir değerden oluşan ve basitçe değiştirilebilen durumlar. (Mesela sayaç gibi)

StateNotifierProvider = Provider'daki **ChangeNotifier**'ın tip-güvenli ve daha düzenli bir versiyonudur. **StateNotifier**, uygulamanın iş mantığını ve verilerini merkezi olarak yönetir, değişiklikleri takip eder ve güvenli bir şekilde diğer bölümlere iletir.

FutureProvider = API'den veri çekme, dosya okuma gibi işlemleri yönetir. Bu işlemler tamamlandığında veriyi ya da oluşan hatayı yakalar. Ayrıca yüklenme (Loading) ve hata (Error) durumlarını otomatik olarak takip eder. Böylece uygulamada ekstra kontrol yazmaya gerek kalmaz.

StreamProvider = Gerçek zamanlı güncellemeler, Websocket (tek bir TCP bağlantısı üzerinden tam çift yönlü iletişim kanalı sağlayan bir bilgisayar iletişim protokolüdür) bağlantıları veya Firebase (mobil uygulamalar ve web uygulamaları için bir geliştirme platformudur) gibi sürekli veri akışı sağlayan işlemler için sıkça kullanılır.

3.3 Neden Riverpod tercih ediliyor?

-Neredeyse tüm hataları (mesela var olmayan bir Provider'ı okumaya çalışmak) uygulama daha çalışmadan, derleme aşamasında yakalar.

-Hangi Provider'ların aktif olarak izlendiğini takip eder. Eğer bir Provider artık kullanılmıyorsa, otomatik olarak bellekten atılabilir. Bu özellikle büyük ve karmaşık uygulamalarda bellek yönetimini optimize eder.

-Widget ağacına bağlı kalmadan uygulamanın durumunu yönetir. Bu da uygulamanın farklı bölümlerinde veri paylaşımını basitleştirir ve kodun daha esnek olmasını sağlar.

3.4 Avantajlar ve Dezavantajlar

Avantajlar

-Hataların çoğu derleme zamanında yakalanır.

-Kaynak temizleme ve en hassas yeniden inşa (*rebuild*) performansı sunar.

Dezavantajlar

-Karmaşık durumlar için biraz daha fazla yapısal kod (**StateNotifier**) gerektirir.

-Provider'dan daha fazla yeni konsept (**ref**) içerir.

4-BloC nedir?

Google tarafından geliştirilen ve uygulamanın iş mantığını kullanıcı arayüzünden ayırarak, tüm veri akışını Olaylar (Events) ve Durumlar (States) aracılığıyla yöneten, değişikliklere hızlı cevap veren ve test edilebilir bir durum yönetimi mimarisidir.

4.1 BloC'un Temel Mantığı

BLoC'un temel mantığı, uygulamanın durum değişimlerini yönetmek için sürekli bir döngü kurmaktır. Bu döngü, verinin sadece tek bir yönde akmasını sağlayan, reaktif bir akış üzerine kuruludur.

BLoC'un varlık sebebi, kullanıcı girdilerini ve çıktılarını bir akış (Stream) aracılığıyla birbirine bağlamaktır. BLoC sınıfı bir giriş noktası (olayları kabul eder) ve bir çıkış noktası (durumları yayınlar) tanımlar.

4.2 BloC Ekosistemi

Cubit<State> = Olaylar yerine doğrudan metot çağrıları ile durumu yöneten BloC'un daha basit bir versiyonudur. **emit(newState)** metodu ile yeni durumu uygulamanın diğer bölümlerine iletir.

MultiBlocProvider = Uygulamanın başlangıcında birden fazla BLoC/Cubit örneğini tek bir yerde tanımlamak için kullanılır.

BlocConsumer<BlocT, StateT> = Hem veriyi gösterip (builder) hem de bir uyarı mesajı göstermek (listener) istendiği zaman kullanılır.

BlocListener<BlocT, StateT> = Hata mesajı gösterme, navigasyon (ekran değiştirme) veya klavye kapatma gibi UI dışı işlemler yapmak için kullanılır.

BlocBuilder<BlocT, StateT> = Sayaç değeri, liste içeriği, metin rengi gibi arayüzün görünümünü değiştirmek için sıkça kullanılır.

4.3 Neden BloC tercih ediliyor?

-Olay-Durum döngüsü, uygulamanın durumunun ne zaman, neden ve nasıl değiştiğine dair kesin bir geçmiş kaydı (log) tutar. Bu, karmaşık hataları bulmayı oldukça kolaylaştırır.

-BLoC, geliştiricileri belirli bir yapıya (Olayları tanımla, Durumları tanımla, Olayları Durumlara eşle) uymaya zorlar. Bu sayede farklı geliştiricilerin yazdığı kodlar arasında yüksek tutarlılık sağlanmış olur.

-BLoC sınıfları, Flutter paketlerine veya UI detaylarına bağımlı değildir. Saf Dart kodudur ve platformdan bağımsızdır. Bu yüzden sık tercih edilir.

4.4 Avantajlar ve Dezavantajlar

Avantajlar

-Hata ayıklaması kolaydır. Durum değişiminin nedeni (Olay) her zaman bellidir.

-Olay => Durum mimarisi büyük projelerde tutarlılığı sağlar.

Dezavantajlar

-Her etkileşim için ayrı **Event** ve **State** sınıfları gerekir.

- Event => State dönüşümü nedeniyle basit etkileşimlerde bile küçük bir gecikme riski oluşturur.

5-Provider, Riverpod, BloC karşılaştırması

5.1 Performans

Provider = Performansı iyi düzeydedir. Ancak, performanslı olması büyük ölçüde geliştiricinin sorumluluğundadır. Geliştiricinin, widget'ın sadece ihtiyaç duyduğu veriyi dinlemesi için **Selector** kullanması gerekir. Aksi takdirde, küçük bir veri değişimi tüm widget'ın gereksiz yere yeniden inşasına yol açabilir.

Riverpod = Üçü arasında en yüksek performansı sunar. **ref.watch()** mekanizması, sadece değişen değerin izlendiği garanti eder. Ek olarak, kullanılmayan Provider'ları otomatik olarak bellekten atarak (kapsülleme), kaynak verimliliğini artırır.

BLoC = Çok iyi performans sergiler. **BlocBuilder** ve **BlocSelector** sayesinde, sadece yeni bir state geldiğinde ve bu state o widget için anlamlıysa yeniden inşa gerçekleşir. Veri akışı katı kurallara bağlı olduğu için hatalı yeniden inşa riski düşüktür.

Özet = Geliştiricinin en az çabayla en yüksek yeniden inşa verimliliğini elde edebileceği çözüm Riverpod'dur. Riverpod, durumu hassas şekilde izleyerek yeniden inşa işlemini neredeyse otomatik olarak optimize eder. BLoC, yapısal kontrolü sayesinde çok iyi bir performans sunarken, Provider ise performansını korumak için geliştiricinin sürekli dikkatli olmasına (manuel Selector kullanımına) ihtiyaç duyar.

5.2 Kod Karmaşıklığı

Provider = En düşük karmaşıklığa sahiptir. Özellikle basit durumlar için sadece bir **ChangeNotifier** sınıfı yeterlidir. Minimal boilerplate kod (projede benzer şekilde tekrar eden otomatikleşmiş ama yazılması gereken temel kalıp kod) gerektirir.

Riverpod = Düşük-Orta karmaşıklıktadır. Basit durumlar için **StateProvider** ile hızlıdır. Karmaşık durumlar için **StateNotifier** kullanılması, kodun ayrışmasını sağlasa da Provider'dan biraz daha fazla boilerplate ekler.

BLoC = En yüksek karmaşıklığa sahiptir. En basit işlem için bile Event, State ve BloC sınıflarının oluşturulması gerekir. Bu yapı, başlangıçta karmaşık ve zaman alıcı görünsede, uzun vadede kodun okunabilirliğini artırır.

Özet = Provider, hızlı başlangıç ve küçük projeler için en az boilerplate kod gerektiren çözümdür. Buna karşın BLoC ise en fazla kod gerektiren çözümdür. Ancak bu karmaşıklık, uzun vadede projenin okunabilirliğini, test edilebilirliğini ve tutarlılığını artırarak yatırımını geri öder. Riverpod ise bu iki yaklaşım arasında dengeli bir yerdedir ve basitlikten ödün vermeden yapısal kaliteyi artırır.

5.3 Öğrenme Eğrisi

Provider = Öğrenme eğrisi en düşüktür. Doğrudan Flutter'ın **InheritedWidget** konsepti üzerine kurulduğu için Flutter geliştiricilerine en tanıdık gelen yapıdır.

Riverpod = Öğrenme eğrisi ortadır. Provider'dan geçiş yapanlar için hızlı, ancak **ref** mekanizması, **ProviderContainer** ve farklı Provider tipleri gibi yeni kavramları öğrenmeyi gerektirir.

BLoC = Öğrenme eğrisi en yüksektir. Reaktif programlama (Stream) ve Events => States dönüşüm felsefesini anlamayı gerektirir. Bu felsefe, geleneksel programlama anlayışından farklıdır.

Özet = Provider doğrudan Flutter'ın yerel yapılarına dayandığı için en kolay çözümdür. Buna karşın BLoC ise reaktif programlama ve Olay => Durum dönüşümü gibi farklı bir felsefe gerektirdiği için öğrenme eğrisi en yüksek olanıdır. Riverpod, Provider'a göre daha fazla soyut kavram içersede, daha fazla güvenlik ve esneklik seçeneğiyle bu iki çözüm arasında dengeli bir konumdadır.

5.4 Ölçeklenebilirlik

Provider = Ölçeklenebilirlik düşük-orta düzeydedir. **BuildContext** bağımlılığı ve Provider'lar arası bağımlılık yönetiminin karmaşıklaşması, büyük projelerde zorluk yaratır. Tip güvenliği eksikliği de büyük projelerde risk taşır.

Riverpod = Ölçeklenebilirlik yüksek düzeydedir. **BuildContext**'ten bağımsızlık, tip güvenliği garantisi ve basit bağımlılık yönetimi **ref.watch()** sayesinde büyük projelerde bile esnekliğini korur.

BLoC = Ölçeklenebilirlik çok yüksek düzeydedir. İş mantığını tam olarak izole eden katı mimarisi, büyük ekiplerin tutarlı kod yazmasını ve tüm iş mantığının %100 test edilebilir olmasını sağlar. Bu yapı, uygulamanın yıllar içinde bile sürdürülebilir kalmasını garantiler

Özet = BLoC, katı mimarisi ve %100 test edilebilirliği sayesinde en yüksek ölçeklenebilirliği sunarken; Riverpod, **BuildContext**'ten bağımsızlığı ve tip güvenliği ile yüksek ölçeklenebilirlik sağlar. Provider ise **BuildContext** bağımlılığı nedeniyle düşük-orta düzeyde kalır.

KAYNAKÇA

Flutter Dev Team. (2024). Flutter Documentation. Google LLC. https://docs.flutter.dev

Remi Rousselet. (2024). Provider Documentation. https://riverpod.dev

Felix Angelov. (2024). Bloc State Management Library. https://bloclibrary.dev

Carmine Zaccagnino. (2020). Programming Flutter. Pragmatic Bookshelf.

Alberto Miola. (2022). *Flutter Complete Reference*. Leanpub.

Marco L. Napoli. (2019). Beginning Flutter: A Hands-On Guide to App Development. Wiley.

Karthik, S. (2022). Provider vs Riverpod vs BLoC: A Practical Comparison. Medium. https://medium.com

Code With Andrea. (2023). Understanding State Management in Flutter. https://codewithandrea.com