Einführung in Riverpod Tutorial

Was ist Riverpod?

Riverpod ist ein State-Management-System für Flutter. Es ermöglicht das Schreiben von flexiblem, testbarem und effizientem Code, indem Zustände und Werte unabhängig definiert und im gesamten Projekt einfach wiederverwendet werden können.

Warum benutzt man nicht einfach globale Variablen?

Globale Variablen mögen auf den ersten Blick praktisch erscheinen, bringen jedoch erhebliche Nachteile mit sich:

- **Schwierige Wartbarkeit**
  Globale Variablen sind überall zugänglich, was zu unübersichtlichem und schwer nachvollziehbarem Code führt. Änderungen an einer globalen Variable können unerwartete Auswirkungen auf den gesamten Code haben.

- **Fehlende Kontrolle über den Lebenszyklus**
  Globale Variablen bleiben während der gesamten Laufzeit der App im Speicher, auch wenn sie nicht mehr benötigt werden. Das erhöht den Speicherverbrauch und erschwert die Verwaltung.

- **Schwierige Testbarkeit**
  Das Testen wird mit globalen Variablen komplex. Es ist schwierig, ihren Zustand für jeden Testfall zu isolieren, was zu instabilen Tests führen kann.

- **Unerwartete Nebenwirkungen**
  Globale Variablen können leicht versehentlich überschrieben oder verändert werden, was zu unvorhersehbarem Verhalten der App führt.

- **Namenskonflikte**
  In großen Projekten kann es leicht passieren, dass Namen von globalen Variablen vergessen oder versehentlich ausgetauscht werden, was zu schwerwiegenden Konflikten führt.

Aus diesen und weiteren Gründen sollte man globale Variablen vermeiden. Stattdessen sollte ein geeignetes State-Management-System verwendet werden.

State-Management in Flutter

In Flutter gibt es verschiedene Methoden für State-Management, darunter:

- **Riverpod**
- **Bloc / Cubit**
- **GetX**
- und mehr.

In diesem Tutorial konzentrieren wir uns ausschließlich auf **Riverpod**.

Warum Riverpod?

Riverpod bietet zahlreiche Vorteile im Vergleich zu anderen State-Management-Lösungen:

1. **Klarheit und Flexibilität**
    - Klare Trennung zwischen Zuständen und deren Abhängigkeiten, was zu modularerem und flexiblerem Code führt.
    - Unterstützt Lazy Initialization, wodurch Ressourcen effizienter genutzt werden.
2. **Einfache Testbarkeit**
    - Riverpod bietet klare und unabhängige Zustandshierarchien, was das Testen erleichtert.
    - Zustände können einfach isoliert und kontrolliert werden, im Gegensatz zu GetX, das stark an den UI-Zustand gebunden sein kann.
3. **Keine BuildContext-Abhängigkeit**
    - Im Gegensatz zu anderen Ansätzen wie Provider benötigt Riverpod keinen BuildContext. Dadurch können Zustände auch außerhalb des Widget-Baums verwaltet werden.
4. **Optimierung von Rebuilds**
    - Riverpod sorgt dafür, dass Widgets nur bei relevanten Änderungen neu aufgebaut werden.
    - Bloc hingegen kann durch zu viele Events komplex werden, was zusätzliche Logik zur Vermeidung unnötiger Rebuilds erfordert.
5. **Abhängigkeitsmanagement**
    - Riverpod nutzt den sogenannten `ProviderScope`, der Abhängigkeiten übersichtlich definiert und kontrolliert.
    - Im Vergleich dazu kann das Dependency-Management bei GetX oder Bloc schnell unübersichtlich werden.
6. **Geringere Lernkurve**
    - Bloc ist zwar sehr mächtig, hat aber eine steilere Lernkurve aufgrund der Event- und State-Architektur.
    - Riverpod ist einfacher zu erlernen und bietet dennoch hohe Flexibilität.
    - GetX ist zwar leicht zu starten, führt aber bei komplexeren Projekten oft zu unkontrolliertem Code (Spaghetti-Code).

Vorgangsweise

Es gibt 12 Applikationen, von denen jede ein spezifisches Konzept erklärt. Dabei wird durch PDF-Datei erläutert, warum dieses Konzept wichtig ist und wie man es umsetzen kann.

Wichtige Hinweise

1. **Ziel des Tutorials**: Das Ziel dieses Tutorials ist es, den Teilnehmern nicht nur zu zeigen, wie das Konzept umgesetzt wird, sondern auch zu erklären, warum dieses Konzept wichtig ist, wann man es anwenden sollte, welche Vorteile und Merkmale es hat und wie man mögliche Nachteile vermeiden kann.
2. **Referenz für Code**
   Dieses Tutorial kann auch als praktische Code-Referenz dienen. Es ist nützlich, wenn Sie beispielsweise nachschlagen möchten, wie man zwei `FutureProvider` in einer Seite verwendet und deren Ergebnisse verarbeitet.
3. **Aufbau des Tutorials**: Das Tutorial besteht aus zwei Teilen: Code und PDF-Erklärung. Jede App enthält eine PDF-Datei, die das jeweilige Konzept des Codes theoretisch erklärt und den Code im Detail verdeutlicht.
4. **Navigation zwischen verschiedenen Applikationen** Wie kann man zwischen die verschiedenen Applikationen navigieren?
   Ich habe eine Klasse `NaviClass` erstellt, die alle Home-Widgets der Applikationen als statische Variablen enthält. Dadurch kann man das gewünschte Widget einfach auswählen und zur main einfügen.

Applikationsziele

1. **App1: StatefulWidget**
   - Erkennen, dass `StatefulWidget` nicht ausreicht und keine Alternative zu einem vollständigen State-Management-System ist.
2. **Installation und Nutzung von Riverpod**
   - Verstehen, wie man Riverpod installiert, im UI verwendet und Providers erstellt.
   - Feststellen, dass Providers nur Werte liefern, aber nicht ändern können.
3. **StateProvider**
   - Aufbau und Funktionsweise des `StateProvider` sowie dessen Nutzung im UI.
4. **StateNotifierProvider**
   - Aufbau und Nutzung des `StateNotifierProvider` im UI.
5. **NotifierProvider**
   - Aufbau und Nutzung des `NotifierProvider` im UI.
6. **FutureProvider**
   - Verwendung des `FutureProvider` für asynchrone Daten.
7. **FutureProvider mit lokalem Dateispeicher**
   - Verwendung des `FutureProvider` in Verbindung mit lokalem Dateispeicher.
8. **AutoDispose Funktionalität**
   - Aufbau und Verständnis der `AutoDispose`-Funktionalität und warum sie wichtig ist.

9. **Family**
    - Aufbau und Nutzung von `Family` im UI.
10. **Family als Filter**

- Aufbau und Anwendung von `Family` als Filter in Riverpod.

11. **Einfaches StreamProvider Beispiel**

- Ein einfaches Beispiel zur Nutzung von `StreamProvider`.

12. **StreamProvider mit lokalem Speicher und zwei Streams**

- Nutzung von `StreamProvider` mit lokalem Speicher und zwei Streams zur Überwachung der Speichersituation, um die Seite nur bei Bedarf neu zu bauen.