**RIVERPOD ARCHITECTURE**

Senza Riverpod: se due widget necessitano di accesso allo stesso stato, quest’ultimo viene passato lungo tutto il widget tree partendo dal widget più alto nella gerarchia fino a raggiungere il widget più in basso che ne necessita 🡪 in parole povere lo stato deve attraversare potenzialmente l’intero widget tree e ciò non è ottimale in molte situazioni portando ad una bassa manutenibilità del codice stesso.

Con Riverpod: se due widget necessitano di accesso allo stesso stato viene definito un “Provider” generale che fornisce lo stato direttamente alla componente interessata, senza attraversare l’intero widget tree. Lo stato viene “innestato” direttamente nella componente interessata.

Immagine che contiene testo, schermata, Carattere, Blu elettrico

Descrizione generata automaticamente VS Immagine che contiene testo, schermata, Carattere, schermo

Descrizione generata automaticamente

Ora aggiornare lo stato significa “raggiungere” il Provider e aggiornare lo stato tramite l’opportuno metodo definito al suo interno. Quando questo avviene, il Provider notifica tutti i widget che consumano lo stato appena mutato, i quali verranno aggiornati di conseguenza.

Riverpod fornisce diversi tipi di Provider, tra cui:

* read-only: consentono la sola operazione di lettura dello stato
* NotifierProvider: scrittura + notifica ai widget dopo il cambiamento
* FutureProvider: per lavorare con dati asincroni (e.g. login con gestione stato autenticato)
* ecc.

Per rendere disponibili i diversi Provider che andiamo a definire inglobiamo l’intera app (main.dart) al ProviderScope. Richiamando l’immagine sopra questo renderà possibile l’accesso ai provider da parte di ogni widget.

Immagine che contiene testo, Carattere, schermata, linea

Descrizione generata automaticamente

Esempio di due read-only Provider:

Immagine che contiene testo, schermata, schermo, software

Descrizione generata automaticamente

Il widget che accede ad uno stato è un widget che consuma lo stato. Riverpod definisce una classe apposita per classificare questa tipologia di widget chiamata *ConsumerWidget*, utilizziamo questa classe quando abbiamo un classico Flutter statelessWidget che vuole consumare uno stato. Quando invece abbiamo un Flutter statefulWidget utilizziamo la controparte offerta da Riverpod chiamata *ConsumerStatefulWidget*.

Esempio di un ConsumerWidget:

Immagine che contiene testo, schermata, Carattere

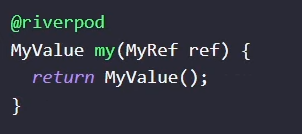
Descrizione generata automaticamente

Quando convertiamo uno StatelessWidget ad un ConsumerWidget ciò che cambia è il secondo parametro nel metodo build() “WidgetRef ref” che possiamo interpretare come una sorta di reference per accedere ai Provider che definiamo nella nostra app. In questo caso, trattandosi di un ConsumerWidget, ciò che ci interessa è leggere lo stato e lo possiamo fare tramite il metodo watch() sulla ref che punta ai Provider definiti. Il metodo watch() consente di osservare lo stato per aggiornare poi il widget se lo stato dovesse cambiare.

Nota: quando codifichiamo un ConsumerStatefulWidget non è necessario dichiarare WidgetRef come attributo del metodo build() poiché già presente in automatico.

**AUTOGENERAZIONE DEI PROVIDER**

Riverpod offre inoltre la possibilità di autogenerare i propri provider rispetto a definirli manualmente sfruttando la seguente sintassi:



Per autogenerare i provider è necessario installare i seguenti pacchetti:

Immagine che contiene testo, Carattere, schermata

Descrizione generata automaticamente

Ora convertiamo i due provider read-only visti precedentemente con la nuova sintassi:

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Nota: Il nome dei provider definiti corrisponde al nome della funzione concatenato con la stringa “Provider” (es. productsProvider, reducedProductsProvider).

Una volta fatto ciò aggiungiamo in testa al file products\_provider.dart, contente i nostri provider, la seguente linea di codice: part ‘products\_provider.g.dart’ che corrisponde al nuovo file che Dart genererà per noi, contenente i nuovi provider. Ora basta eseguire da terminale il comando: *dart run build\_runner watch* per autogenerare il file dichiarato in precedenza. Basta eseguirlo una sola volta per progetto e poi ogni volta che useremo la nuova sintassi per autogenerare provider basterà salvare e i provider saranno disponibili.

Nota: l’autogenerazione dei provider vede il suo pieno utilizzo in casi decisamente più complessi di quello sopra riportato in cui la differenza tra la definizione manuale e automatica non è evidente; tuttavia, utilizzarli fin da subito per l’intero progetto è una buona practice.

**NOTIFIER PROVIDER**

Consente di aggiornare lo stato di un provider e notificare il cambiamento a tutti i widget che consumano quello stato.

Per definire un NotifierProvider definiamo una nuova classe che estende il tipo “Notifier<T>”, dobbiamo inoltre dichiarare il tipo T che corrisponde al tipo di oggetto su cui il Provider lavora e il tipo di oggetto ritornato ai widget che consumano lo stato.

Nel nostro caso vogliamo definire uno stato che gestisce i prodotti inseriti nel carrello quindi il tipo T sara “Set<Product>”.

All’interno della classe dobbiamo definire:

* un valore iniziale dello stato ritornato dal metodo build() alla prima costruzione;
* metodi che consento di alterare lo stato.

Nello stesso file, dopo aver definito la classe sopra citata, definiamo il Provider vero e proprio che ritorna un’istanza della classe appena creata in modo da esporla a tutti i widget che la consumano. Qui sotto un esempio semplice del tutto.

Immagine che contiene testo, schermata, schermo, software

Descrizione generata automaticamente

Proviamo ora a generare il NotifierProvider utilizzando il tool offerto da Riverpod:

Immagine che contiene testo, schermata, software

Descrizione generata automaticamente

**DEPENDENT PROVIDER**

Provider che dipendono da altri Provider. Seguendo il nostro esempio, vogliamo creare un Provider che calcoli il prezzo totale degli articoli nel carrello. Naturalmente questo Provider dipende dagli articoli contenuti nel carrello, esposti da cartNotifierProvider.

Immagine che contiene testo, schermata, Carattere

Descrizione generata automaticamente

Il nuovo Provider osserva cartNotifierProvider e aggiorna il suo valore ogni qual volta lo stato degli articoli nel carrello cambia.