Flutter e Dart - Le basi

Flutter - Widgets

Luigi Durso

luigi.durso@si2001.it

SI2001

25 giugno 2022

Sommario

Un po' di codice

Analizziamo l'elaborato precedente!

Layers

Come prima cosa individuiamo i layers:

- **Data layer**: Si occupa dell'interazione diretta con delle API per il recupero dati;
- **Domain layer**: Layer responsabile della ricezione, manipolazione e trasformazione dei dati (services, repositories, DTOs);
- **Business logic**: Gestione dello stato dell'applicativo (vedremo in seguito);
- **Presentation**: Renderizzare le componenti della UI in base allo stato;

Scegliere la struttura da seguire

Due approcci consigliati:

- Layer First;
- Features Firts

Layer first

```
--lib
   |--screens
   |--widgets
   |--services
   |--view_models
   |--services
   |-- ...
```

Figura: Layer first example [1]

Struttura layer first

Come intuibile dal nome, questo tipo di struttura si basa su una suddivisione dell'intero progetto in cartelle che contengono i vari componenti di ogni layer;

Feature first

```
feature1/
├ domain/
│ ├ models/
│ │ └ feature1_model.dart
│ ├ repository/
│ │ └ feature1_repository.dart
│ ├ services/
│ │ └ feature1_service.dart
│ └ feature1_domain.dart
├ providers/
│ ├ feature1_provider.dart
│ └ providers.dart
├ screens/
│ ├ feature1_screen.dart
│ └ screens.dart
├ widgets/
│ ├ feature1_widget.dart
│ └ widgets.dart
└ index.dart
```

Figura: Feature first example [1]

Struttura feature first

Questo approccio è adatto a progetti grandi, scala molto bene mantenendo una struttura pulita e organizzata. Il concetto base è la creazione di una cartella per ogni funzionalità dell'applicativo. Come si può notare dall'immagine, ogni cartella conterrà un mini progetto in "Layer First".

La base di ogni applicativo in Flutter: il **Widget**

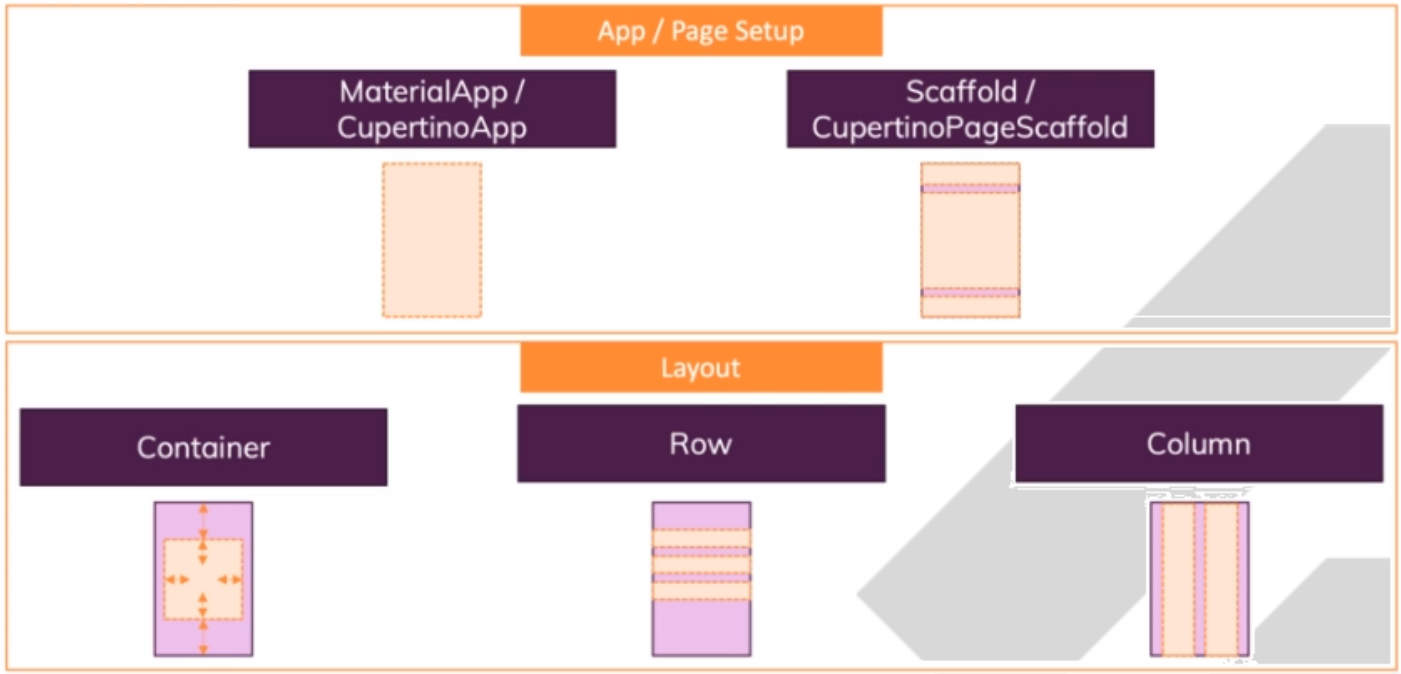
Possiamo individuarne diversi tipi:

- Widget per l'impostazione di Applicazione o Pagina;
- Widget di Layout;
- Widget wrapper per contenuti;
- Widget per la renderizzazione di liste di elementi;
- Widget per l'input da parte dell'utente...

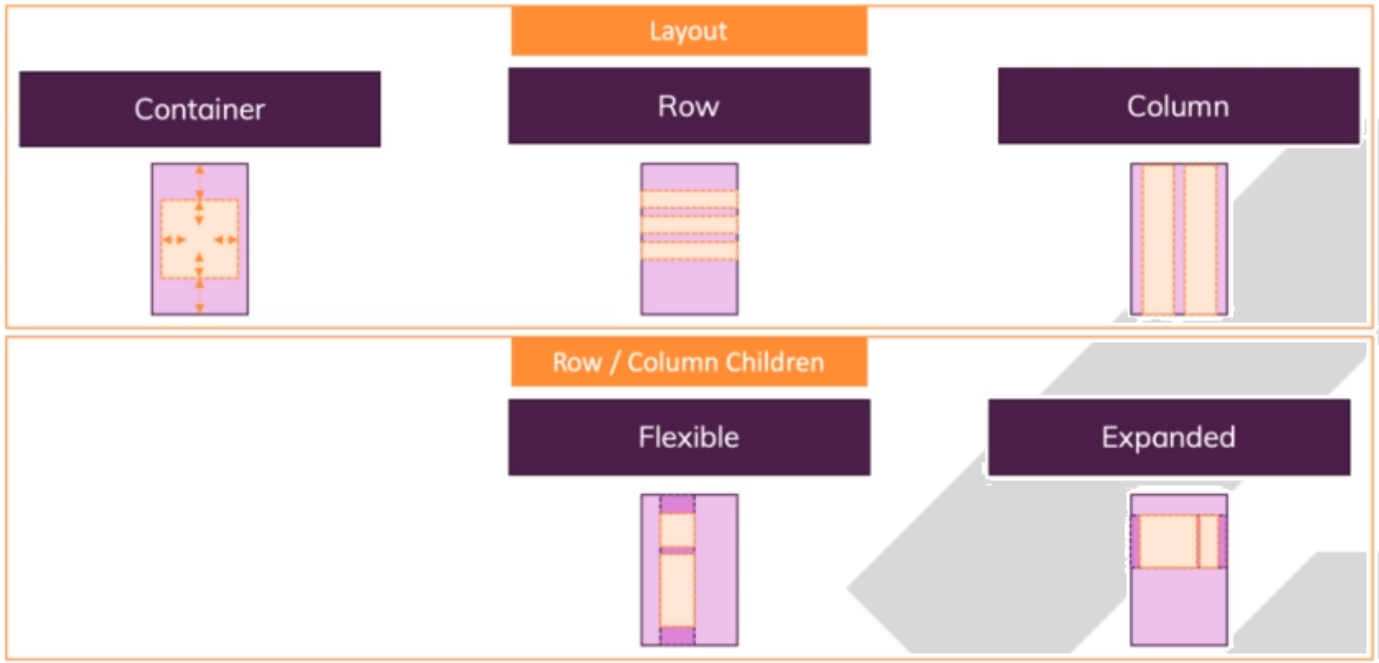
Facciamo sempre riferimento al catalogo

▸ Catologo

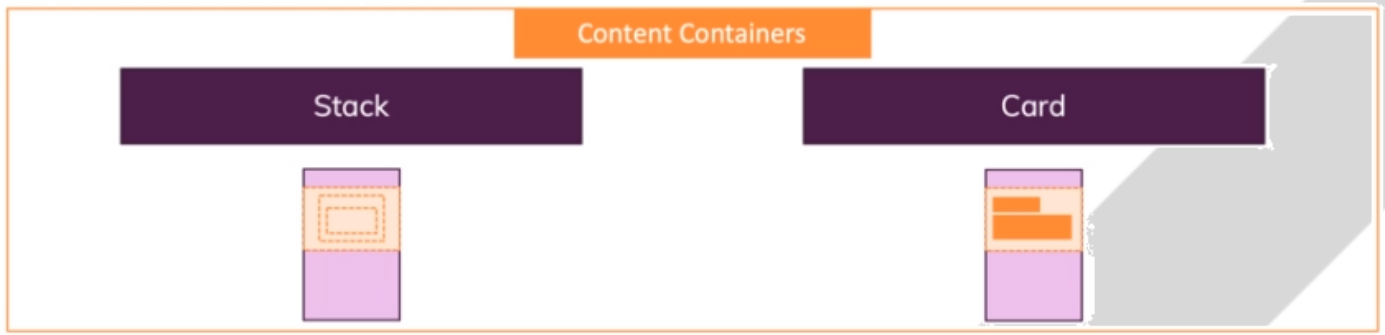
Categorie di widgets



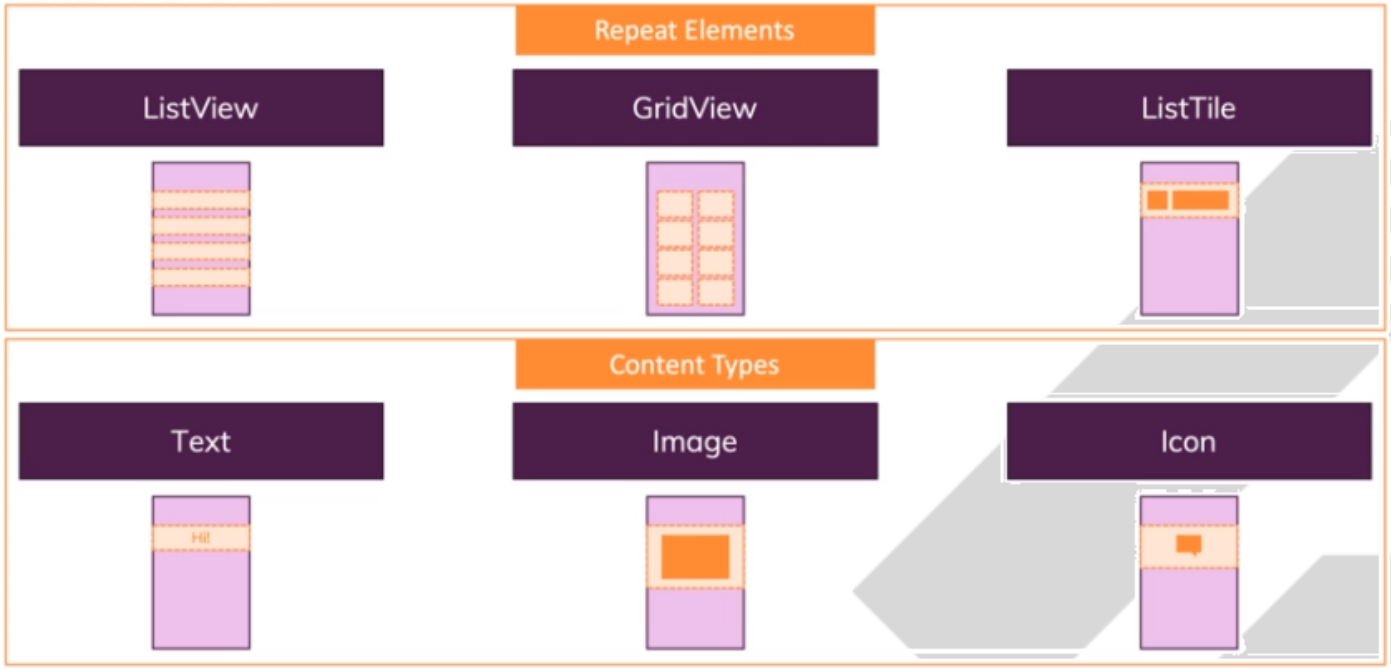
Categorie di widgets



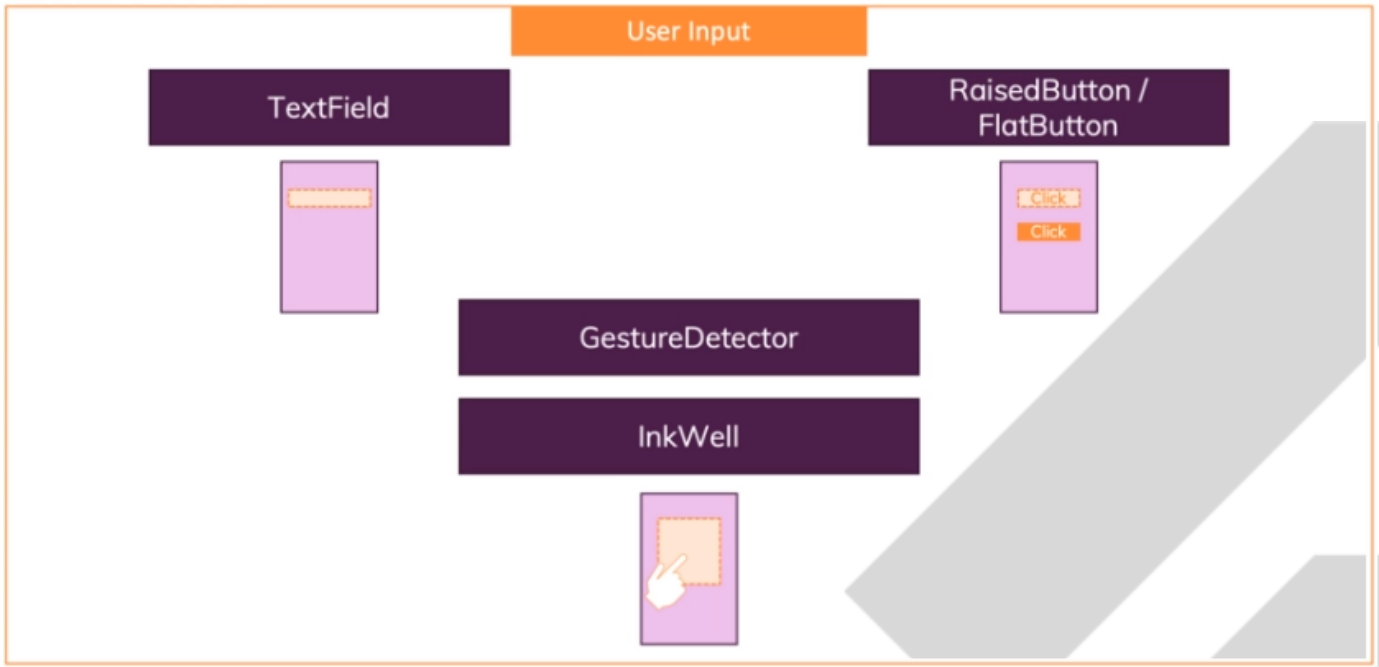
Categorie di widgets



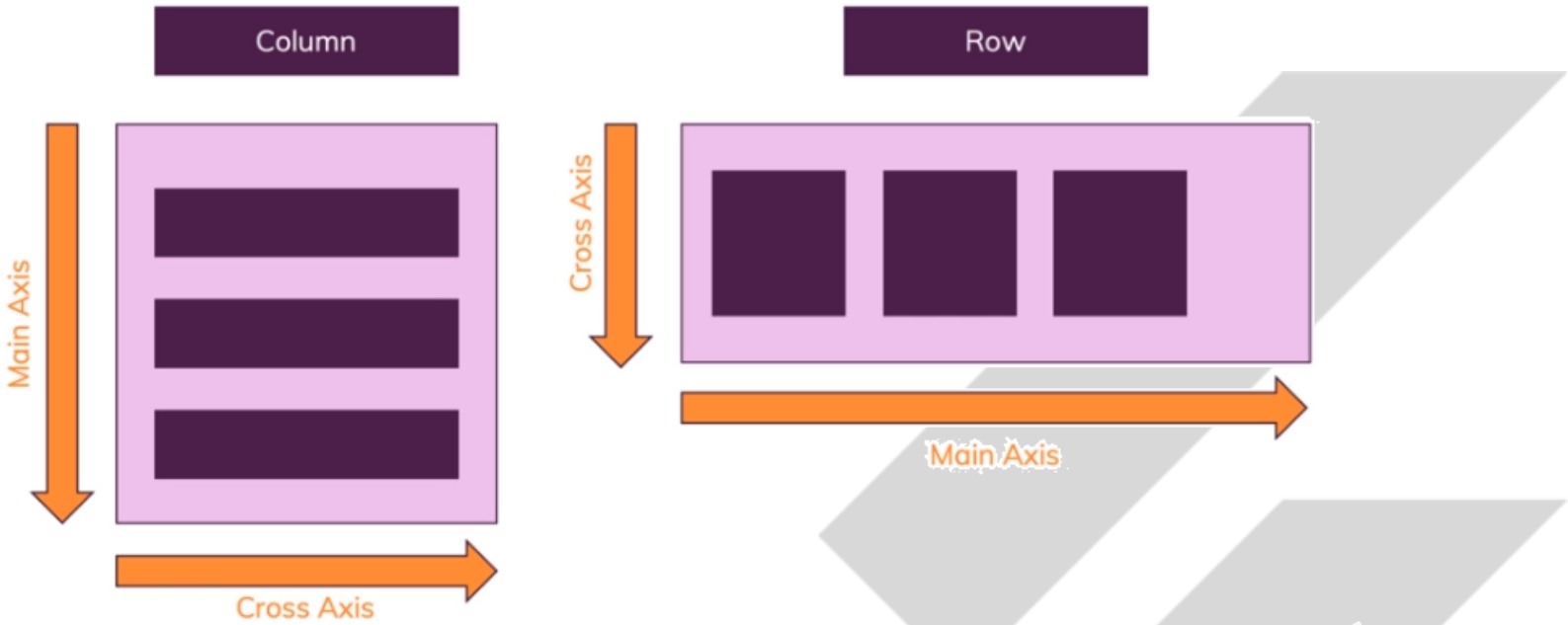
Categorie di widgets



Categorie di widgets



Widgets colonna e riga



Combinare i widget

Non sempre si trova il widget perfetto per il caso d'uso. Analizzali singolarmente e combinali.

Un esempio:

Container

- Riceve esattamente un figlio;
- Molte opzioni per lo stile
- Perfetto se cerchiamo alta customizzazione

Colonna/Riga

- Riceve figli multipli;
- Nessuna opzione di stile
- Prendono sempre tutto lo spazio disponibile nella loro main direction
- Perfetto se cerchiamo un allineamento tra più widget

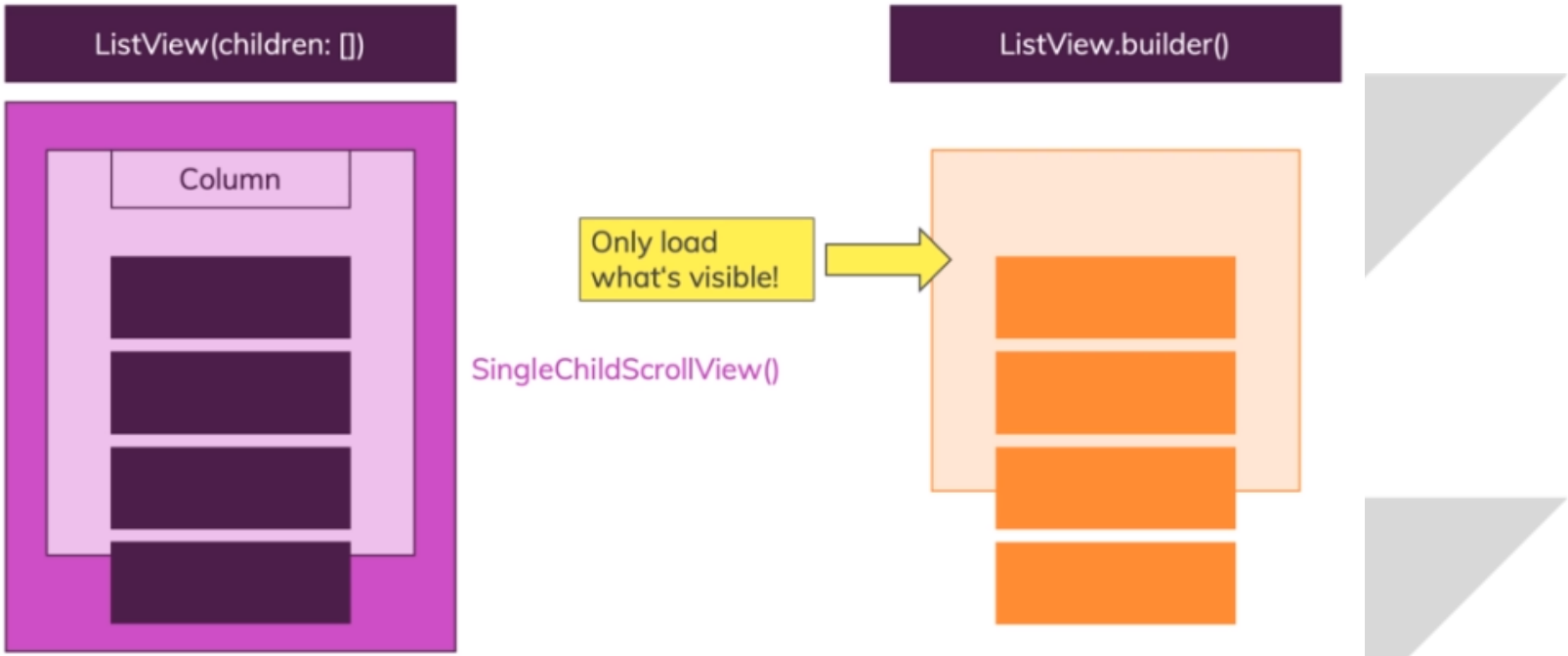
Lavorare con le liste

Per renderizzare elementi multipli possiamo considerare due widgets:

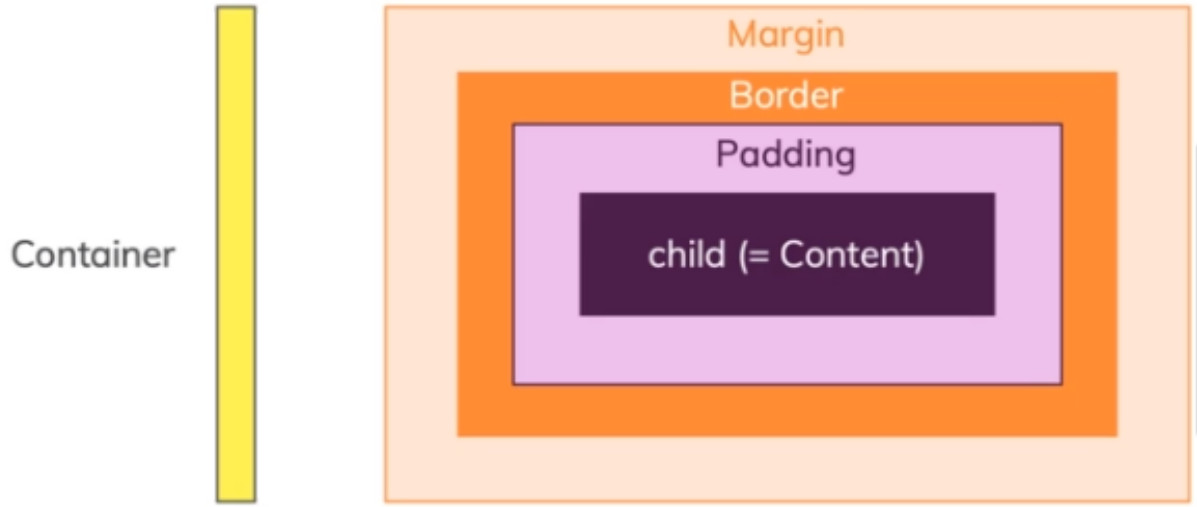
- ListView;
- GridView;

Renderizzazione liste

ListView e GridView hanno due modalità di utilizzo:



Per le distanze valgono le stesse regole del CSS



Customizzare lo stile

Possiamo applicare lo stile in due modi:

- Ogni widget accetta lo stile nel costruttore;
- Creare un tema globale per l'applicativo, la maggior parte dei widget faranno riferimento ad esso per gli elementi contenuti;

Come sempre, fare riferimento al catalogo

▸ Catalogo

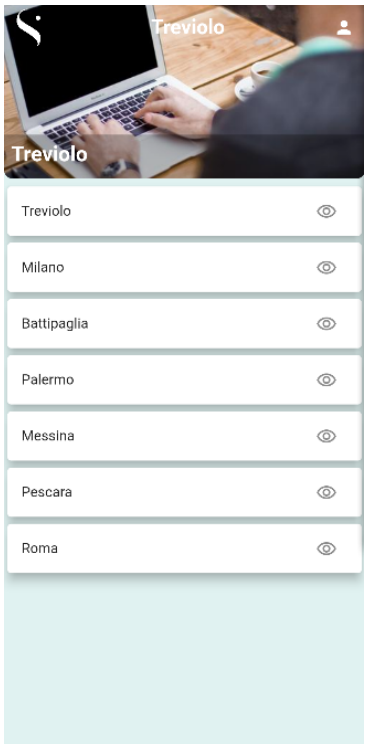
Tema globale

Utilizzare un tema globale:

- Come linea guida di uno stile uniforme in tutta l'applicazione;
- I widget seguono per le proprie caratteristiche lo stile configurato nel tema;

Documentazione

Migliorare la precedente esercitazione



Alcuni spunti:

- Creare un tema globale per l'app;
- Utilizzare ListView e ListTile
- Creare un'area in cima che visualizzi la sede selezionata, inserire bordi arrotondati nella parte inferiore
- ogni elemento avrà un'immagine presa dal web, inserirla come sfondo nella parte superiore;

Riferimenti I

 Ryan Dsilva. **Scalable Folder Structure for Flutter Applications**. 2022.

Grazie per l'attenzione!