

Programmazione Mobile

Nicola Noviello

nicola.noviello@unimol.it

Corso di Laurea in Informatica
Dipartimento di Bioscienze e Territorio
Università degli Studi del Molise
Anno 2023/2024

Lezione: Flutter e Dart - Funzionalità avanzate (parte prima)

- Rendering dei contenuti in base a criteri condizionali
- Approfondimento del ciclo di vita di un Widget Stateful
- Espressioni ternarie e operatori condizionali
- Data Model
- Stili, Allineamento, Margini e Padding
- Integrazione con Package di terze parti
- Contenuti Scrollabili



Realizziamo un'App
per quiz e sondaggi

Il progetto di partenza

```
lib > main.dart > ...
1   import 'package:flutter/material.dart';
2   import 'package:lezione7/start_screen.dart';
3
4   void main() {
5       runApp(
6           MaterialApp(
7               home: Scaffold(
8                   body: Container(
9                       decoration: const BoxDecoration(
10                          gradient: LinearGradient(
11                              begin: Alignment.bottomRight,
12                              end: Alignment.topLeft,
13                              colors: [Colors.white, Color.fromARGB(255, 22, 60, 118)],
14                          ), // LinearGradient
15                          ), // BoxDecoration
16                          child: const StartScreen(),
17                      ), // Container
18                  ), // Scaffold
19              ), // MaterialApp
20          );
21      }◆;
22  |
```

Il progetto di partenza

```
2
3 class StartScreen extends StatelessWidget {
4   const StartScreen({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return Center(
9       child: Column(
10      mainAxisAlignment: MainAxisAlignment.center,
11      children: [
12        Image.asset('assets/unimol_logo.png', width: 300.0),
13        const SizedBox(height: 40.0),
14        const Text(
15          'App Unimol per i Quiz!',
16          style: TextStyle(
17            fontSize: 24.0,
18            //           fontWeight: FontWeight.bold,
19            color: Colors.white,
20          ), // TextStyle
21        ), // Text
22        const SizedBox(height: 60.0),
23        OutlinedButton(
24          onPressed: () {},
25          style: OutlinedButton.styleFrom(
26            shape: RoundedRectangleBorder(
27              borderRadius: BorderRadius.circular(20.0),
28            ), // RoundedRectangleBorder
29            side: const BorderSide(color: Colors.white),
30            foregroundColor: Colors.white,
31          ),
32          child: const Text('Inizia a giocare'),
33        ), // OutlinedButton
34      ],
35    ), // Column
```

Il progetto di partenza





Si comincia

OutlinedButton e Costruttori Multipli

Voglio aggiungere un'icona, quali opzioni ho?

```
2
3 class StartScreen extends StatelessWidget {
4   const StartScreen({super.key});
5
6   @override
7   Widget build(BuildContext context) {
8     return Center(
9       child: Column(
10         mainAxisAlignment: MainAxisAlignment.center,
11         children: [
12           Image.asset('assets/unimol_logo.png', width: 300.0),
13           const SizedBox(height: 40.0),
14           const Text(
15             'App Unimol per i Quiz!',
16             style: TextStyle(
17               fontSize: 24.0,
18               // fontWeight: FontWeight.bold,
19               color: Colors.white,
20             ), // TextStyle
21           ), // Text
22           const SizedBox(height: 60.0),
23           OutlinedButton(
24             onPressed: () {},
25             style: OutlinedButton.styleFrom(
26               shape: RoundedRectangleBorder(
27                 borderRadius: BorderRadius.circular(20.0),
28               ), // RoundedRectangleBorder
29               side: const BorderSide(color: Colors.white),
30               foregroundColor: Colors.white,
31             ),
32             child: const Text('Inizia a giocare'),
33           ), // OutlinedButton
34         ],
35       ), // Column
36     );
37 }
```

OutlinedButton e Costruttori Multipli

factory viene utilizzato per creare un meccanismo di creazione alternativo per una classe, consentendo la creazione di istanze della classe in modo non necessariamente legato al costruttore standard. Quando si definisce un metodo **factory** all'interno di una classe, questo metodo è responsabile della creazione di nuove istanze della classe e può restituire istanze della classe stesse o di sottoclassi

```
Open online interactive samples for OutlinedButton
class OutlinedButton extends ButtonStyleButton {
    /// Create an OutlinedButton.
    const OutlinedButton({
        super.key,
        required super.onPressed,
        super.onLongPress,
        super.onHover,
        super.onFocusChange,
        super.style,
        super.focusNode,
        super.autofocus = false,
        super.clipBehavior = Clip.none,
        super.statesController,
        required super.child,
    });

    /// Create a text button from a pair of widgets that serve as the button's
    /// [icon] and [label].
    ///
    /// The icon and label are arranged in a row and padded by 12 logical pixels
    /// at the start, and 16 at the end, with an 8 pixel gap in between.
    factory OutlinedButton.icon({
        Key? key,
        required VoidCallback? onPressed,
        VoidCallback? onLongPress,
        ButtonStyle? style,
```

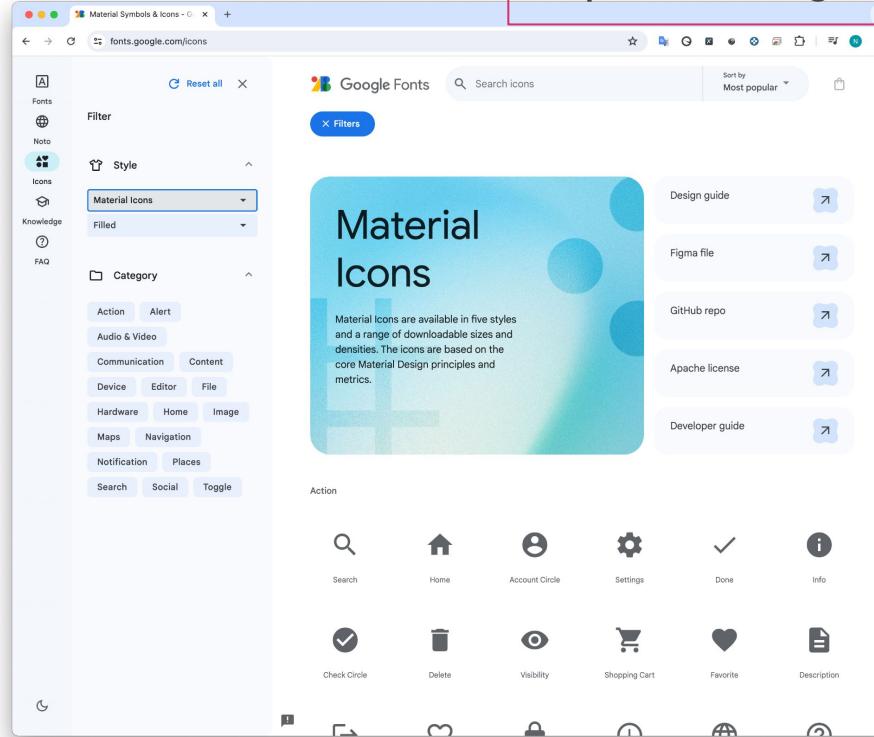
OutlinedButton.icon

```
    ),
    OutlinedButton.icon(
        onPressed: () {},
        icon: const Icon(Icons.play_arrow),
        style: OutlinedButton.styleFrom(
            shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(20.0),
            ), // RoundedRectangleBorder
            side: const BorderSide(color: Colors.white),
            foregroundColor: Colors.white,
        ),
        label: const Text('Inizia a giocare!'),
    ), // OutlinedButton.icon
```



Material Icons

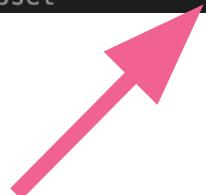
<https://fonts.google.com/icons>



Integriamo trasparenza nell'immagine



```
Image.asset(  
  'assets/logomonocromo.png',  
  width: 300.0,  
  color: Color.fromRGBO(120, 255, 255, 255),  
) // Image.asset
```



Integriamo trasparenza nell'immagine

```
Image.asset(  
    'assets/logomonocromo.png',  
    width: 300.0,  
    color: Color.fromRGBO(120, 201, 32, 32),  
) // Image.asset
```

La nostra immagine però non è
adattabile a questa tecnica 😢



Integriamo trasparenza nell'immagine

```
Opacity(  
  opacity: 0.5,  
  child: Image.asset('assets/unimol_logo.png', width: 300.0),  
) // Opacity
```

Per un'immagine non fa niente, ma lavorare con il canale alpha in tempo reale è tremendamente esoso in termini di risorse 💀 quindi bisogna scegliere con criterio le immagini da utilizzare





PENSATI
STATEFUL

Creiamo un nuovo Widget StatefulWidget per la pagina che conterrà le domande

Usiamo sempre i plugin

```
lib > ⚙️ questions_screen.dart > [?] stf
1  stf
2  ⚡ Flutter StatefulWidget
3  ⚡ StackFilter
4  ⚡ StackFit
5  ⚡ StackFit
6  ⚡ StackFit
    ⚡ StackFit
    ⚡ StackFrame
    ⚡ StandardFabLocation
    ⚡ StatefulWidget
    ⚡ StatefulWidget
    ⚡ StatefulWidget
    ⚡ StatefulWidget
    ⚡ StatefulWidget
```

Creiamo un nuovo Widget StatefulWidget per la pagina che conterrà le domande

Al momento usiamo solo un Placeholder

```
lib > questions_screen.dart > ...
1   import 'package:flutter/material.dart';
2
3   class QuestionsWidget extends StatefulWidget {
4     const QuestionsWidget({super.key});
5
6     @override
7     State<QuestionsWidget> createState() => _QuestionsWidgetState();
8   }
9
10  class _QuestionsWidgetState extends State<QuestionsWidget> {
11    @override
12    Widget build(BuildContext context) {
13      return const Placeholder();
14    }
15  }
16
17
```

Portiamo il blocco principale in un Widget StatefulWidget

Perché? La nostra App non ha più dei contenuti statici...

```
lib > first_screen.dart > ...
1   import 'package:flutter/material.dart';
2   import 'package:lezione7/start_screen.dart';
3
4   class FirstScreen extends StatefulWidget {
5     const FirstScreen({super.key});
6
7     @override
8     State<FirstScreen> createState() => _FirstScreenState();
9   }
10
11  class _FirstScreenState extends State<FirstScreen> {
12    @override
13    Widget build(BuildContext context) {
14      return MaterialApp(
15        home: SafeArea(
16          child: Scaffold(
17            body: Container(
18              decoration: const BoxDecoration(
19                gradient: LinearGradient(
20                  begin: Alignment.bottomRight,
21                  end: Alignment.topLeft,
22                  colors: [Colors.white, Color.fromARGB(255, 22, 60, 118)],
23                ), // LinearGradient
24                ), // BoxDecoration
25                child: const StartScreen(),
26              ), // Container
27            ), // Scaffold
28          ), // SafeArea
29        ); // MaterialApp
30      }
31    }◆
32  |
```

Portiamo il blocco principale in un Widget Stateful

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:lezione7/first_screen.dart';
3
4 void main() {
5   runApp(
6     const FirstScreen(),
7   );
8 }
9
```

Una volta adeguati gli import sembra comunque non cambiare nulla. Perché? Cosa manca?

Definiamo i Widget per cambiare schermata

```
class _FirstScreenState extends State<FirstScreen> {  
    var activeScreen = const StartScreen();  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            home: SafeArea(  
                child: Scaffold(  
                    body: Container(  
                        decoration: const BoxDecoration(  
                            gradient: LinearGradient(  
                                begin: Alignment.bottomRight,  
                                end: Alignment.topLeft,  
                                colors: [Colors.white, Color.fromARGB(255, 22, 60, 118)],  
                            ), // LinearGradient  
                        ), // BoxDecoration  
                        child: activeScreen,  
                    ), // Container  
                ), // Scaffold  
            ), // MaterialApp  
    } // build  
}
```



Definiamo i Widget per cambiare schermata

```
class _FirstScreenState extends State<FirstScreen> {
    var activeScreen = const StartScreen();

    void switchNewScreen() {
        setState(() {
            activeScreen = const QuestionsWidget();
        });
    }
}
```

Dovrebbe funzionare, perché questo errore?

Definiamo i Widget per cambiare schermata

```
use
FirstScreen> A value of type 'QuestionsWidget' can't be assigned to a variable of type 'StartScreen'.
Try changing the type of the variable, or casting the right-hand type to
'StartScreen'. dart(invalid\_assignment)
FirstScreenState
  StartScreen startScreen = QuestionsWidget QuestionsWidget({Key? key})
  void switchNewScreen() {
    View Problem (⌘F8) Quick Fix... (⌘.)
    StartScreen startScreen = const QuestionsWidget();
```

Siamo stati troppo restrittivi...

Definiamo i Widget per cambiare schermata

```
class _FirstScreenState extends State<FirstScreen> {
    Widget activeScreen = const StartScreen();

    void switchNewScreen() {
        setState(() {
            activeScreen = const QuestionsWidget();
        });
    }
}
```

Abbiamo implementato il **Rendering condizionale**
di un contenuto

Situazione attuale

StartScreen

Pagina di benvenuto con
immagine e pulsante

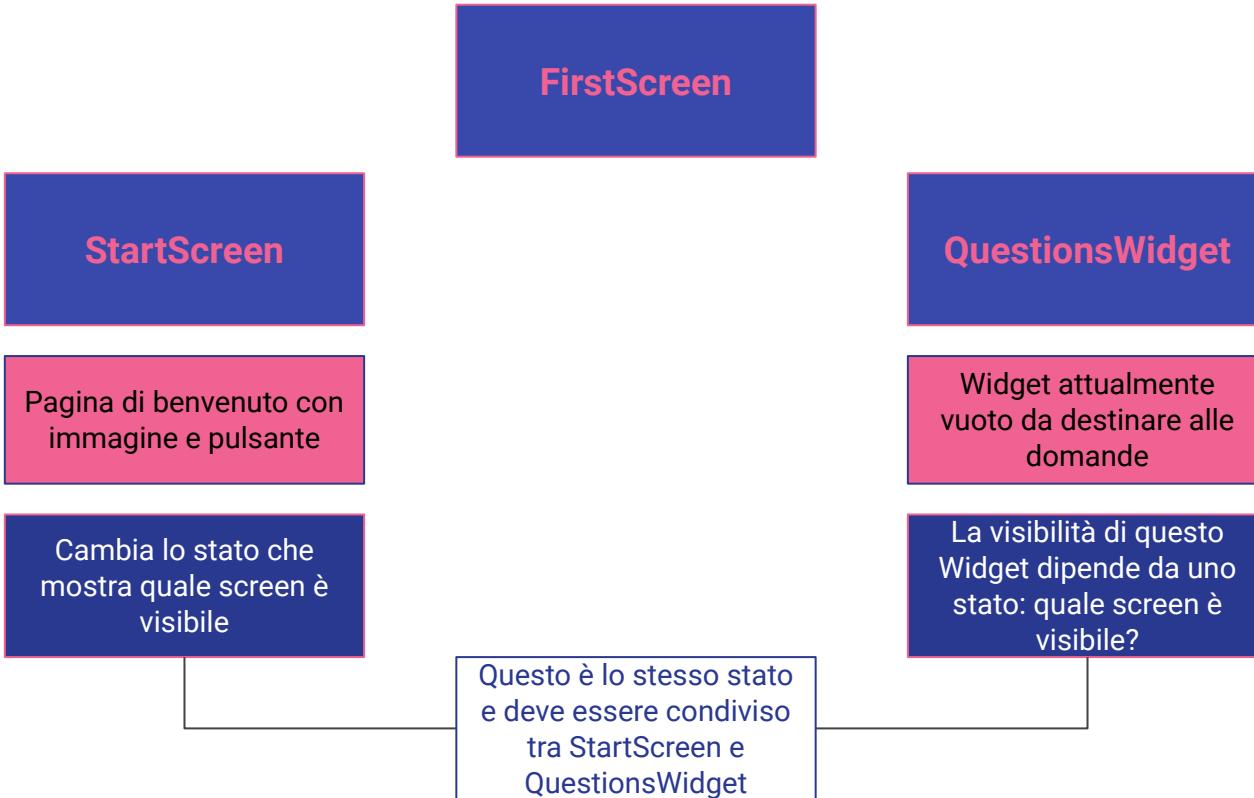
FirstScreen

Il main.dart trasformato
in StatefulWidget che di
fatto ha solo un
Container con gradiente

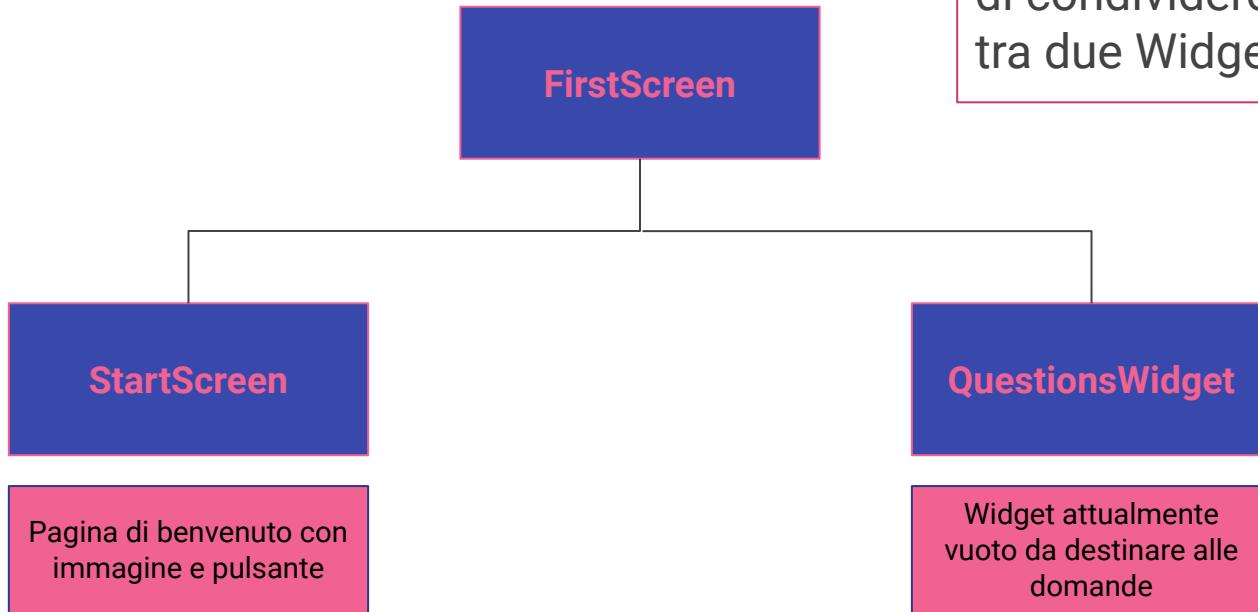
QuestionsWidget

Widget attualmente
vuoto da destinare alle
domande

Situazione attuale



Lifting Up State - Design Pattern



Andiamo a creare un Widget “padre” in grado di condividere lo stato tra due Widget figli

Definiamo i Widget per cambiare schermata

```
class _FirstScreenState extends State<FirstScreen> {
    Widget activeScreen = const StartScreen();

    void switchNewScreen() {
        setState(() {
            activeScreen = const QuestionsWidget();
        });
    }
}
```

Quindi quello che manca è capire come chiamare switchNewScreen() per cambiare effettivamente schermata

Switch tra schermate

```
main.dart
first_screen.dart 3
start_screen.dart
questions_screen.dart
logomonocromo.png
outlined_button.dart

> first_screen.dart > _FirstScreenState > switchNewScreen()
  import 'package:flutter/material.dart';
  import 'package:lezione7/questions_screen.dart';
  import 'package:lezione7/start_screen.dart';

  class FirstScreen extends StatefulWidget {
    const FirstScreen({super.key});

    @override
    State<FirstScreen> createState() => _FirstScreenState();
  }

  class _FirstScreenState extends State<FirstScreen> {
    Widget activeScreen = const StartScreen(switchNewScreen);

    void switchNewScreen() {
      setState(() {
        activeScreen = const QuestionsWidget();
      });
    }
  }
}

Too many positional arguments: 0 expected, but 1 found.
Try removing the extra positional arguments, or specifying the name for named
arguments. dart(extra_positional_arguments_could_be_named)

Arguments of a constant creation must be constant expressions.
Try making the argument a valid constant, or use 'new' to call the
constructor. dart(const_with_non_constant_argument)

The instance member 'switchNewScreen' can't be accessed in an initializer.
Try replacing the reference to the instance member with a different
expression dart(implicit_this_reference_in_initializer)

void switchNewScreen()
  package:lezione7/first_screen.dart
```

Funzione privata - non definita la funzione nel costruttore

Switch tra schermate

```
lib > start_screen.dart > StartScreen > build
1   import 'package:flutter/material.dart';
2
3   class StartScreen extends StatelessWidget {
4     const StartScreen(this.switchScreen, {super.key});
5     final Function() switchScreen;
6
7     @override
8     Widget build(BuildContext context) {
9       return Center(
10       child: Column(
11         mainAxisAlignment: MainAxisAlignment.center,
12         children: [
13           Opacity(
14             opacity: 0.5,
15             child: Image.asset('assets/unimol_logo.png', width: 300.0),
16           ), // Opacity
17           // Image.asset(
18           //   'assets/logomonocromo.png',
19           //   width: 300.0,
20           //   color: Color.fromARGB(120, 255, 255, 255),
21           // ),
22           const SizedBox(height: 40.0),
23           const Text(
24             'App Unimol per i Quiz!',
25             style: TextStyle(
26               fontSize: 24.0,
27               // fontWeight: FontWeight.bold,
28               color: Colors.white,
29             ), // TextStyle
30           ), // Text
31           const SizedBox(height: 60.0),
32           OutlinedButton.icon(
33             onPressed: switchScreen,
34             icon: const Icon(Icons.play_arrow),
```

Switch tra schermate

```
class FirstScreen extends StatefulWidget {
  const FirstScreen({super.key});

  @override
  State<FirstScreen> createState() =>
}

class _FirstScreenState extends State<FirstScreen> {
  Widget activeScreen = StartScreen(switchNewScreen);

  void switchNewScreen() {
    setState(() {
      activeScreen = const QuestionsWidget();
    });
  }
}
```

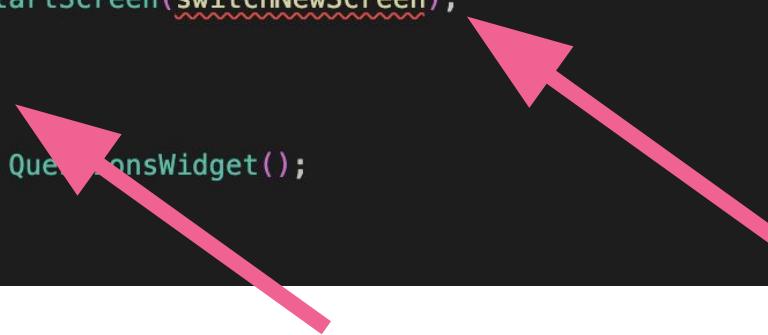
The instance member 'switchNewScreen' can't be accessed in an initializer.
Try replacing the reference to the instance member with a different
expression dart([implicit_this_reference_in_initializer](#))

View Problem (⌘F8) Quick Fix... (⌘.)

initState

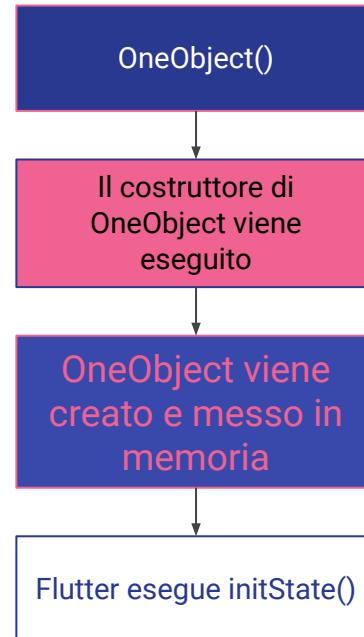
```
class _FirstScreenState extends State<FirstScreen> {
    Widget activeScreen = StartScreen(switchNewScreen);

    void switchNewScreen() {
        setState(() {
            activeScreen = const QuestionsWidget();
        });
    }
}
```



Vengono eseguiti in contemporanea. Dobbiamo usare quindi un “workaround”

Usare initState()



Inizializzo la classe

Variabili e metodi vengono
creati

Posso usare ulteriori task di
inizializzazione

initState

```
class _FirstScreenState extends State<FirstScreen> {
    Widget? activeScreen;

    @override
    void initState() {
        activeScreen = StartScreen(switchNewScreen);
        super.initState();
    }

    void switchNewScreen() {
        setState(() {
            activeScreen = const QuestionsWidget();
        });
    }

    @override
```

Ciclo di vita dei Widget Stateful

Ogni Widget di Flutter ha un ciclo di vita integrato: una serie di metodi eseguiti automaticamente da Flutter (in determinati momenti).

Ci sono tre metodi estremamente importanti del ciclo di vita di un **Widget Stateful**:

- **initState()**: Eseguito da Flutter quando lo State object del **Widget Stateful** viene **inizializzato**
- **build()**: Eseguito da Flutter quando il Widget viene costruito per la **prima volta** e ogni altra volta quando viene chiamato il metodo **setState()**
- **dispose()**: Eseguito da Flutter proprio prima che il Widget venga eliminato (ad es., perché è stato visualizzato condizionalmente)

Operatore Ternario

```
class _FirstScreenState extends State<FirstScreen> {
    // ...
    // void initState() {
    //     activeScreen = StartScreen(switchNewScreen);
    //     super.initState();
    // }
    var activeScreen = 'start-screen';
    void switchNewScreen() {
        setState(() {
            activeScreen = 'questions-screen';
        });
    }
}

@Override
Widget build(BuildContext context) {
    return MaterialApp(
        home: SafeArea(
            child: Scaffold(
                body: Container(
                    decoration: const BoxDecoration(
                        gradient: LinearGradient(
                            begin: Alignment.bottomRight,
                            end: Alignment.topLeft,
                            colors: [Colors.white, Color.fromARGB(255, 22, 60, 118)],
                        ), // LinearGradient
                    ), // BoxDecoration
                    // da usare in caso di initState
                    // child: activeScreen,
                    child: activeScreen == "start-screen"
                        ? StartScreen(switchNewScreen)
                        : const QuestionsWidget(),
                ), // Container
            ),
        ),
    );
}
```

Esercizio 1

La funzionalità implementata con un semplice if può essere anche fatta dall'operatore ternario. Scaricate lo zip con il nome esercizio1, provate il codice visto a lezione ed implementate in autonomia una vostra soluzione utilizzando l'if

Uso degli operatori per la costruzione dinamica di liste

```
1 void main() {  
2   bool condition = true;  
3   final myList = [1, 2, if (condition) 3];  
4   print(myList);  
5 }  
6
```



Run

[1, 2, 3]

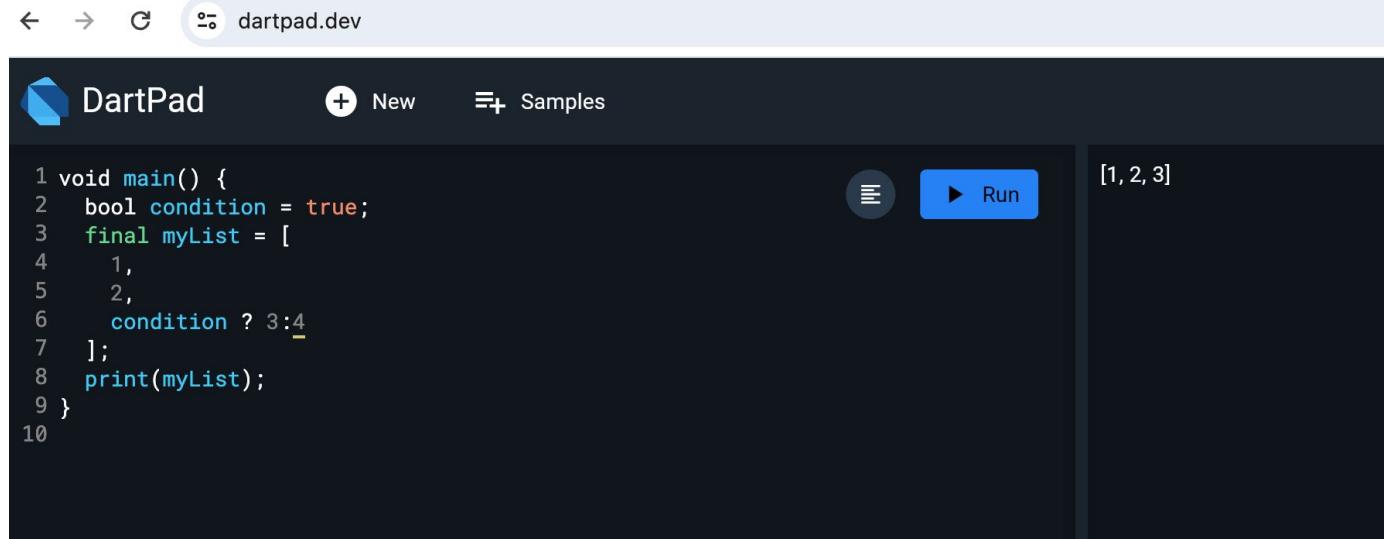
Uso degli operatori per la costruzione dinamica di liste

```
1 void main() {  
2     bool condition = false;  
3     final myList = [  
4         1,  
5         2,  
6         if (condition)  
7             3  
8         else  
9             4  
10    ];  
11    print(myList);  
12 }  
13
```



[1, 2, 4]

Uso degli operatori per la costruzione dinamica di liste



The screenshot shows a DartPad interface. The top bar has back, forward, and refresh buttons, and a search field containing "dartpad.dev". The main area is titled "DartPad" with "New" and "Samples" buttons. On the left, there's a code editor with the following Dart code:

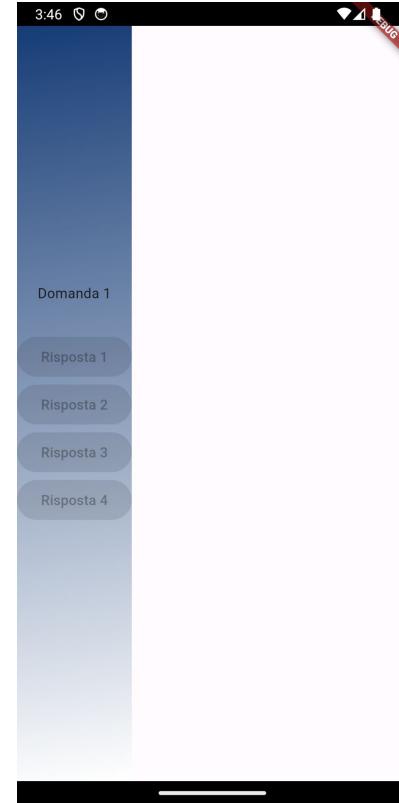
```
1 void main() {
2   bool condition = true;
3   final myList = [
4     1,
5     2,
6     condition ? 3:4
7   ];
8   print(myList);
9 }
10
```

On the right, there are two panes: one with a "Run" button and another showing the output "[1, 2, 3]".

Documentazione sull'uso di if e for per costruire collection

Struttura di QuestionsWidget

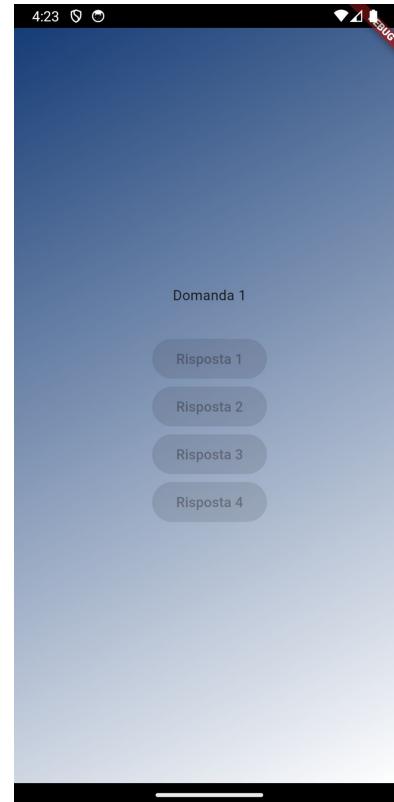
```
class _QuestionsWidgetState extends State<QuestionsWidget> {
  @override
  Widget build(BuildContext context) {
    return const Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text('Domanda 1'),
        SizedBox(height: 30.0),
        ElevatedButton(
          onPressed: null,
          child: Text('Risposta 1'),
        ), // ElevatedButton
        ElevatedButton(
          onPressed: null,
          child: Text('Risposta 2'),
        ), // ElevatedButton
        ElevatedButton(
          onPressed: null,
          child: Text('Risposta 3'),
        ), // ElevatedButton
        ElevatedButton(
          onPressed: null,
          child: Text('Risposta 4'),
        ), // ElevatedButton
      ],
    ); // Column
  }
}
```



Struttura di QuestionsWidget

```
class _QuestionsWidgetState extends State<QuestionsWidget> {
    @override
    Widget build(BuildContext context) {
        return const SizedBox(
            width: double.infinity, // allargamento massimo
            child: Column( // Column ...
        ); // SizedBox
    }
}
```

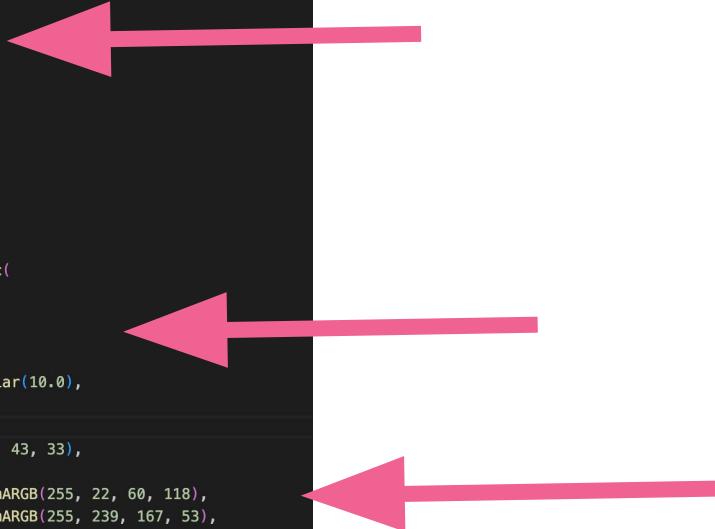
Un'alternativa al Center



Astrazione del bottone delle risposte

Ovviamente da un'altra parte dobbiamo usare AnswerButton() al posto di ElevatedButton...

```
lib > answer_button.dart > AnswerButton > build
1 import 'package:flutter/material.dart';
2
3 class AnswerButton extends StatelessWidget {
4   const AnswerButton({
5     super.key,
6     required this.answerText,
7     required this.onTap,
8   });
9   final String answerText;
10  final void Function() onTap;
11  @override
12  Widget build(BuildContext context) {
13    return ElevatedButton(
14      onPressed: onTap,
15      style: ElevatedButton.styleFrom(
16        padding: const EdgeInsets.symmetric(
17          vertical: 10.0,
18          horizontal: 50.0,
19        ), // EdgeInsets.symmetric
20        shape: RoundedRectangleBorder(
21          borderRadius: BorderRadius.circular(10.0),
22        ), // RoundedRectangleBorder
23        side: const BorderSide(
24          color: Color.fromARGB(255, 176, 43, 33),
25        ), // BorderSide
26        backgroundColor: const Color.fromARGB(255, 22, 60, 118),
27        foregroundColor: const Color.fromARGB(255, 239, 167, 53),
28      ),
29      child: Text(answerText),
30    ); // ElevatedButton
31  }
32}
33
```





Parliamo di dati

Data Model

The screenshot shows a dark-themed code editor interface. On the left, the Explorer sidebar displays project files under 'LEZIONE7', including '.dart_tool', '.idea', 'android', 'assets', 'build', 'ios', 'lib', and 'models'. Inside 'models', there is a file named 'question.dart'. The main editor area shows the code for the 'Question' class:

```
1  class Question {  
2      final String question;  
3      final List<String> answers;  
4  
5      Question(this.question, this.answers);  
6 }◆◆◆  
7
```

The code editor has tabs for 'main.dart', 'questions_screen.dart', and 'question.dart', with 'question.dart' currently selected. There are also icons for file operations like creating, deleting, and renaming files.

Data

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar:
 - LEZIONE7
 - .dart_tool
 - .idea
 - android
 - assets
 - build
 - ios
 - lib
 - data
 - questions.dart
 - models
 - question.dart
 - first_screen.dart
 - main.dart
 - questions_screen.dart
 - start_screen.dart
- ... button

The **questions.dart** file is selected in the Explorer, and its content is displayed in the main editor area.

questions.dart content:

```
lib > data > questions.dart > questions
1 import 'package:lezione7/models/question.dart';
2
3 const questions = [
4   Question(
5     'Quali sono i principali blocchi di costruzione delle UI di Flutter?',
6     [
7       'Widgets',
8       'Componenti',
9       'Blocchi',
10      'Funzioni',
11    ],
12  ),
13  Question('Come vengono costruite le UI di Flutter?', [
14    'Combinando widget nel codice',
15    'Combinando widget in un editor visuale',
16    'Definendo widget nei file di configurazione',
17    'Utilizzando XCode per iOS e Android Studio per Android',
18  ]),
19  Question('Cosa sono i blocchi di costruzione delle UI di Flutter?')]
```

Integrazione dei dati in maniera statica

```
lib > lib/questions_screen.dart > _QuestionsWidgetState > build
  1 import 'package:flutter/material.dart';
  2 import 'package:lezione7/answer_button.dart';
  3 import 'package:lezione7/data/questions.dart';
  4
  5 class QuestionsWidget extends StatefulWidget {
  6   const QuestionsWidget({super.key});
  7
  8   @override
  9   State<QuestionsWidget> createState() => _QuestionsWidgetState();
 10 }
 11
 12 class _QuestionsWidgetState extends State<QuestionsWidget> {
 13   @override
 14   Widget build(BuildContext context) {
 15     final currentQuestion = questions[0];
 16     return SizedBox(
 17       width: double.infinity, // allargamento massimo
 18       child: Column(
 19         mainAxisAlignment: MainAxisAlignment.center,
 20         children: [
 21           Text(
 22             currentQuestion.question,
 23             style: const TextStyle(
 24               fontSize: 15.0,
 25               fontWeight: FontWeight.bold,
 26               color: Colors.white), // TextStyle
 27               textAlign: TextAlign.center,
 28             ), // Text
 29             const SizedBox(height: 30.0),
 30             AnswerButton(
 31               answerText: currentQuestion.answers[0],
 32               onTap: () {},
 33             ), // AnswerButton
 34             AnswerButton(
 35               answerText: currentQuestion.answers[1],
 36               onTap: () {},
 37             ), // AnswerButton
 38             AnswerButton(
 39               answerText: currentQuestion.answers[2],
 40               onTap: () {}
```



Iteratori

```
23         style: const TextStyle(
24             fontSize: 15.0,
25             fontWeight: FontWeight.bold,
26             color: Colors.white), // TextStyle
27             textAlign: TextAlign.center,
28         ), // Text
29         const SizedBox(height: 30.0),
30         currentQuestion.answers
31         .map((answer) => AnswerButton(
32             answerText: answer,
33             onTap: () {},
34         )), // AnswerButton
35     > // AnswerButton(
36     ],
37     ), // Column
38 ); // SizedBox
39 }
40 }
```

Iteratori

```
20       children: [
21         Text(
22           currentQuestion.question,
23           style: const TextStyle(
24             The element type 'Iterable<AnswerButton>' can't be assigned to the list type
25             'Widget'. dart(list_element_type_not_assignable)
26
27           String answer
28
29           Type: String
30           View Problem (F8) Quick Fix... (%)
31           .map((answer) => AnswerButton(
32             answerText: answer,
33             onTap: () {},
34           )), // AnswerButton
35         // AnswerButton(
36         ],
37       ), // Column
38     ); // SizedBox
39   }
40 }
```

Restituisce un Iterable, ma noi abbiamo bisogno di Widget, dobbiamo adottare quindi una tecnica che viene definita spreading

Spreading Values

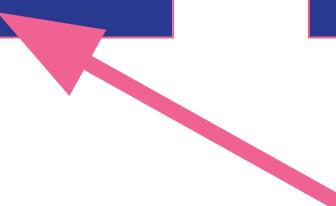
```
const numbers = [ 1,2,3];
```

```
const moreNum = [numbers, 4];
```

```
[ [1,2,3], 4 ]
```

```
const moreNum = [...numbers, 4];
```

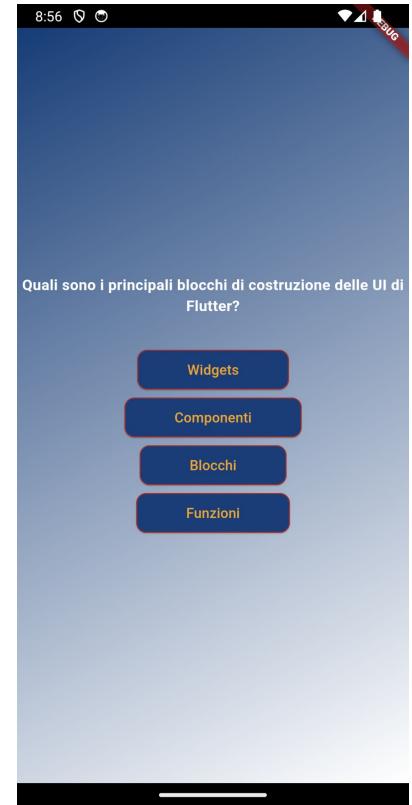
```
[ 1,2,3,4 ]
```



operatore Spread

Spread

```
), // text  
const SizedBox(height: 30.0),  
...currentQuestion.answers  
    .map((answer) => AnswerButton(  
        answerText: answer,  
        onTap: () {},  
    )), // AnswerButton  
> // AnswerButton(...  
],
```



Esercizio 2

Scaricate lo zip con il nome esercizio2, provate a trasformare il quiz testuale in un quiz con delle immagini come risposta

Margini tra gli elementi

```
class _QuestionsWidgetState extends State<QuestionsWidget> {  
  @override  
  Widget build(BuildContext context) {  
    final currentQuestion = questions[0];  
    return SizedBox(  
      width: double.infinity, // allargamento marginato  
      child: Container(  
        margin: const EdgeInsets.all(40.0),  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          crossAxisAlignment: CrossAxisAlignment.stretch,  
          children: [  
            Text(  
              currentQuestion.question,  
              style: const TextStyle(  
                fontSize: 15.0,  
                fontWeight: FontWeight.bold,  
                color: Colors.white), // TextStyle  
              textAlign: TextAlign.center,  
            ), // Text  
            const SizedBox(height: 30.0),  
            ...currentQuestion.answers.map((answer) => AnswerButton(  
              answerText: answer,  
              onTap: () {},  
            )) // AnswerButton  
          ],  
        ),  
      ),  
    );  
  }  
}
```

Abbiamo impostato il margine al di fuori della colonna e poi abbiamo allargato fino ai bordi i singoli pulsanti



Padding vs Margin

Padding

- Il **Padding** è lo spazio interno aggiunto intorno al contenuto di un widget
- Si utilizza per inserire spazio tra il contenuto del widget e i suoi bordi. Ad esempio, se si desidera aggiungere spazio tra il testo all'interno di un pulsante e i bordi del pulsante stesso, si applica il **Padding**

Margin

- Il **Margin** è lo spazio esterno aggiunto attorno al bordo di un widget
- Si utilizza per creare spazio tra il widget e gli elementi circostanti. Ad esempio, se si desidera spaziare un widget rispetto agli altri elementi nell'interfaccia utente, si applica il **Margin**

Mischio le domande, funzione shuffle()

```
models/question.dart -> Question -> getShuffledAnswers  
class Question {  
    final String question;  
    final List<String> answers;  
  
    const Question(this.question, this.answers);  
    List<String> getShuffledAnswers() {  
        final shuffledAnswers = List.of(answers); // copia, a noi serve l'originale  
        shuffledAnswers  
            .shuffle(); // funziona con final perché non cambia la lista in memoria  
        return shuffledAnswers;  
    }  
}
```

Mischio le domande, funzione shuffle()

```
        fontWeight: FontWeight.bold,
        color: Colors.white), // TextStyle
        textAlign: TextAlign.center,
    ), // Text
    const SizedBox(height: 30.0),
    ...currentQuestion
        .getShuffledAnswers()
        .map((answer) => AnswerButton(
            answerText: answer,
            onTap: () {}),
        )), // AnswerButton
    // AnswerButton(
    //     answerText: currentQuestion.answers[0],
    //     onTap: () {},
    // ),
    // AnswerButton(

```

Integro una funzione in grado di cambiare domanda

```
class _QuestionsWidgetState extends State<QuestionsWidget> {
    var currentQuestionIndex = 0;
    void answerQuestion() {
        setState(() {
            currentQuestionIndex++;
        });
    }
}

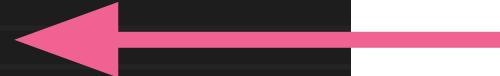
@Override
Widget build(BuildContext context) {
    final currentQuestion = questions[currentQuestionIndex];
    return SizedBox(
        width: double.infinity, // allargamento massimo
        child: Container(

```

Integro una funzione in grado di cambiare domanda

```
        fontSize: 15.0,
        fontWeight: FontWeight.bold,
        color: Colors.white), // TextStyle
        textAlign: TextAlign.center,
    ), // Text
    const SizedBox(height: 30.0),
    ...currentQuestion
        .getShuffledAnswers()
        .map((answer) => AnswerButton(
            answerText: answer,
            onTap: answerQuestion,
        )), // AnswerButton
    // AnswerButton/

```



```
17 | );
18 |
19 |
20 |
21 @override
22 Widget build(BuildContext context) {
    final currentQuestion = questions[currentQuestionIndex];
    // ...
}

Exception has occurred. ×
RangeError (RangeError (index)): Invalid value: Not in inclusive range 0..15: 16

23 | return SizedBox(
24 |   width: double.infinity, // allargamento massimo
25 |   child: Container(
26 |     margin: const EdgeInsets.all(40.0),

```

E se volessimo cambiare un font?

The screenshot shows a web browser displaying the Flutter documentation at docs.flutter.dev/cookbook/design/fonts. The page title is "Use a custom font". The left sidebar contains a navigation menu with categories like User interface, Design & theming, Text, Interactivity, Assets & media, Navigation & routing, Animations & transitions, Accessibility & internationalization, Beyond UI, Data & backend, Platform integration, Packages & plugins, Testing & debugging, Performance & optimization, and Deployment. Under the "Text" category, the "Use a custom font" link is highlighted. The main content area starts with a section titled "Contents" which includes links to "Choose a font", "Import the font files", and "Declare the font in the pubspec.yaml file". Below this is a section titled "What you'll learn" with a bulleted list: "How to choose a font.", "How to import font files.", "How to set a font as a default.", and "How to use a font in a given widget.". At the bottom of the page, there is a note: "Although Android and iOS offer high quality system fonts, designers want support for custom fonts. You might have a custom-built font from a designer, or perhaps you downloaded a font from [Google Fonts](#). A typeface is the collection of glyphs or shapes that comprise a given style of lettering. A font is one representation of that typeface at a given weight or variation. Roboto is a typeface and Roboto Bold is a font." Further down, it says: "Flutter lets you apply a custom font across an entire app or to individual widgets. This recipe creates an app that uses custom fonts with the following steps." followed by a numbered list: 1. Choose your fonts., 2. Import the font files., 3. Declare the font in the pubspec., 4. Set a font as the default., 5. Use a font in a specific widget. At the very bottom, it says: "You don't need to follow each step as you go. The guide offers completed example files at the end."

C'è una soluzione più pratica

Integrare package esterni - pub.dev

The screenshot shows the details of the `google_fonts` package on the `pub.dev` website. The package version is 6.2.1, published 59 days ago, and is compatible with Dart 3. It has 5.5K dependencies. The package has 5547 likes, 130 pub points, and 100% popularity. The `Readme` tab is selected, showing the package's purpose: "A Flutter package to use fonts from fonts.google.com." Below the description is a screenshot of a Flutter application displaying the text "google_fonts demo" in various Google Fonts. The `Example` tab shows the source code for the application:

```
11   return MaterialApp(
12     debugShowCheckedModeBanner: if the [style]
13     home: Scaffold(
14       appBar: AppBar(
15         title: Text(
16           'google_fonts demo',
17           style: GoogleFonts[
18             'Google Sans',
19             style: TextStyle(
20               color: Colors.pink,
21             ),
22           ],
23         ),
24       ),
25     );
26   } // Scaffold
27 } // MaterialApp
28 
```

The `Versions` tab lists the available versions: 6.2.1, 6.1.0, 6.0.0, 5.0.0, 4.0.0, 3.0.0, 2.0.0, 1.0.0, and 0.0.1. The `Scores` tab is also visible.

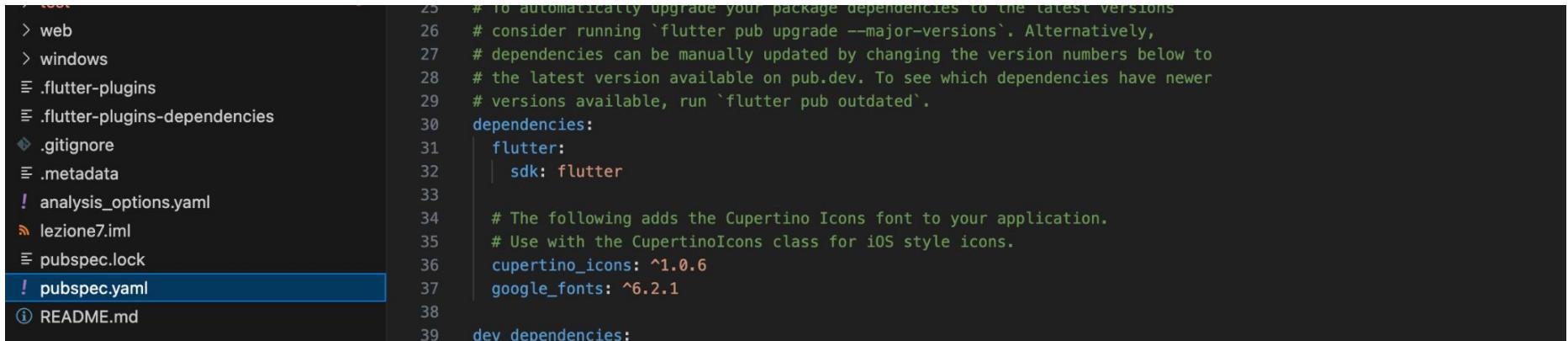
https://pub.dev/packages/google_fonts

Installazione

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

● nico@Mac-mini-di-Nico lezione7 % flutter pub add google_fonts
Resolving dependencies...
+ crypto 3.0.3
+ ffi 2.1.2
+ google_fonts 6.2.1
+ http 1.2.1
+ http_parser 4.0.2
leak_tracker 10.0.0 (10.0.5 available)
leak_tracker_flutter_testing 2.0.1 (3.0.5 available)
leak_tracker_testing 2.0.1 (3.0.1 available)
material_color_utilities 0.8.0 (0.11.1 available)
meta 1.11.0 (1.14.0 available)
+ path_provider 2.1.3
+ path_provider_android 2.2.4
+ path_provider.foundation 2.3.2
+ path_provider_linux 2.2.1
+ path_provider_platform_interface 2.1.2
+ path_provider_windows 2.2.1
+ platform 3.1.4
+ plugin_platform_interface 2.1.8
test_api 0.6.1 (0.7.1 available)
+ typed_data 1.3.2
vm_service 13.0.0 (14.2.1 available)
+ web 0.5.1
+ win32 5.5.0
+ xdg_directories 1.0.4
Changed 17 dependencies!
7 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
○ nico@Mac-mini-di-Nico lezione7 % █
```

Installazione



```
  test
> web
> windows
Ξ .flutter-plugins
Ξ .flutter-plugins-dependencies
◆ .gitignore
Ξ .metadata
! analysis_options.yaml
↳ lezione7.iml
Ξ pubspec.lock
! pubspec.yaml
ⓘ README.md
```

```
25  # To automatically upgrade your package dependencies to the latest versions
26  # consider running `flutter pub upgrade --major-versions`. Alternatively,
27  # dependencies can be manually updated by changing the version numbers below to
28  # the latest version available on pub.dev. To see which dependencies have newer
29  # versions available, run `flutter pub outdated`.
30  dependencies:
31    flutter:
32      |  sdk: flutter
33
34  # The following adds the Cupertino Icons font to your application.
35  # Use with the CupertinoIcons class for iOS style icons.
36  cupertino_icons: ^1.0.6
37  google_fonts: ^6.2.1
38
39  dev_dependencies:
```

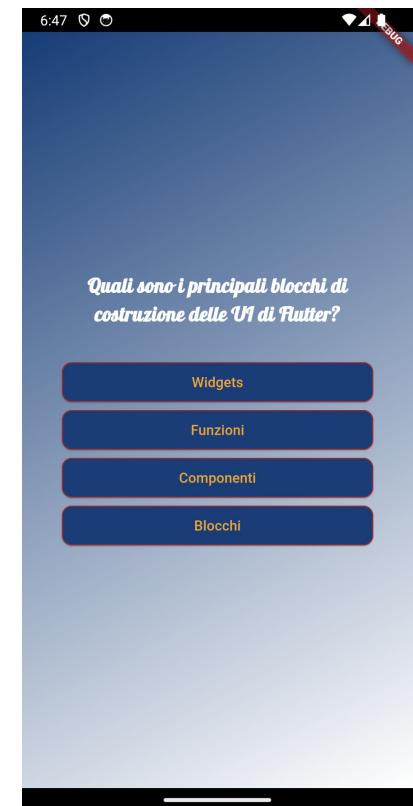
Utilizzo

```
import 'package:tezione/data/questions.dart';
import 'package:google_fonts/google_fonts.dart';

> class QuestionsWidget extends StatefulWidget { ...

class _QuestionsWidgetState extends State<QuestionsWidget> {
  var currentQuestionIndex = 0;
  void answerQuestion() {
    setState(() {
      currentQuestionIndex++;
    });
  }

  @override
  Widget build(BuildContext context) {
    final currentQuestion = questions[currentQuestionIndex];
    return SizedBox(
      width: double.infinity, // allargamento massimo
      child: Container(
        margin: const EdgeInsets.all(40.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          crossAxisAlignment: CrossAxisAlignment.stretch,
          children: [
            Text(
              currentQuestion.question,
              style: GoogleFonts.lobster(
                fontSize: 20.0,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
            ),
          ],
        ),
      ),
    );
  }
}
```



Esercizio 3

Esplorate <https://pub.dev> ed integrate uno o due packages a vostra scelta