BCSCHOOL

Empowering Digital Skills For The Jobs Of The Future

BC Soft
consulenza sistemi informativi

by

AiRED
Artificial Intelligence Ricerca e Sviluppo

Interazioni, stili, animazioni

Docente

Claudia Infante

claudia.infante@bcsoft.net

Widget di stile

Il widget MaterialApp fornisce una serie di vantaggi per la personalizzazione dell'applicazione. Aggiunge le funzionalità e il design system di Material design con le relative specifiche e funzionalità legate allo stile.

Widget di stile

Scaffold è il widget principale dedicato alla struttura dell'applicazione. Insieme a MaterialApp definisce il layout visivo.

Widget di stile

Prevede le seguenti proprietà :

Key key	appBar	body
floatingActionButton	loatingActionButtonLocation	loatingActionButtonAnimator
persistentFooterButtons	drawer	endDrawer
bottomNavigationBar	bottomSheet	backgroundColor
resizeToAvoidBottomPadding	primary	

Stili e temi

Il widget Theme definisce lo stile di defalut dell'app come i colori e il font, applicati automaticamente all'intera applicazione. È accessibile ovunque e può essere sovrascritto.

Stili e temi

Alcune proprietà sono brightness, primarySwatch, primaryColor e accentColor nonché proprietà specifiche come dividerColor, buttonColor, errorColor.

Stili e temi

Per esempio Theme.primaryColor influenza l'intera applicazione cambiando i colori dei widget in base al colore definito.

Stili e temi

La classe per configurare un tema è **ThemeData.** Per aggiungere un tema all'applicazione si passa un oggetto ThemeData alla proprietà theme di MaterialApp.

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
                colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}
```

Stili e temi

Le proprietà che i widget utilizzano sono ereditate del tema widget più vicino. È possibile dunque creare più temi nell'applicazione che sovrascriveranno il principale.

Stili e temi

L’unità di misura utilizzata è il pixel logico. Per strutturare un applicazione in modo programmatico bisogna utilizzare il MediaQuery widget.

Stili e temi

MediaQuery è un widget simile a Theme.

È possibile accedervi in qualsiasi punto dell'applicazione utilizzando BuildContext attraverso il metodo of della classe MediaQuery.

Stili e temi

Il metodo of cerca nell'albero, trova la classe MediaQuery più vicina e fornisce l'accesso.

Per fare riferimento all'istanza di MediaQuery in qualsiasi punto dell'applicazione, si può utilizzare il metodo 'of' fornito da alcuni dei widget integrati in Flutter.

Stili e temi

Il metodo MediaQuery.of(context).size ritorna le informazioni sulla dimensione dello schermo del device. Una volta acquisite le informazioni, possono essere utilizzate per determinare la dimensione del widget in base alla larghezza e all'altezza dello schermo.

Stili e temi

Per esempio, per ottenere le dimensioni dell'80% dello schermo si utilizzerà :

```
final width = MediaQuery.of(context).size.width * 0.8;
```

Stili e temi

```dart
@override
  Widget build(BuildContext context) {
    // Get the screen size using MediaQuery
    final screenSize = MediaQuery.of(context).size;

    Color containerColor = screenSize.width < 700 ? Colors.green.shade400 : Colors.blue;

    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('MediaQuery Example'),
        ), // AppBar
        body: Center(
          child: Container(
            width: screenSize.width * 0.8,
            height: screenSize.height * 0.4,
            color: containerColor,
            child: Text(
              'Screen Width: ${screenSize.width}\nScreen Height: ${screenSize.height}',
              style: const TextStyle(
                color: Colors.white,
                fontSize: 20,
              ), // TextStyle
            ), // Text
          ), // Container
        ), // Center
      ), // Scaffold
    ); // MaterialApp
  }
}
```

Stili e temi

Lo **Stack** widget è invece utilizzato per posizionare gli elementi gli uni sugli altri. Per posizionare i widget figli si utilizzerà il widget Positioned il quale prevede le proprietà top, left, right, bottom, width e height che indicano a Flutter dove posizionare gli elementi.

Stili e temi

```
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Stack Example'),
        ), // AppBar
        body: Stack(
          children: [
            Container(
              color: Colors.blue,
            ), // Container
            Positioned(
              top: 50,
              left: 50,
              child: Container(
                width: 200,
                height: 200,
                color: Colors.red,
              ), // Container
            ), // Positioned
            const  Positioned(
                top: 100,
                left: 100,
                child: Text(
                  'Hello, World!',
                  style: TextStyle(
                    fontSize: 24,
                    fontWeight: FontWeight.bold,
                    color: Colors.white,
                  ), // TextStyle
                ), // Text
              ), // Positioned
            ],
          ), // Stack
        ), // Scaffold
      ); // MaterialApp
    }
}
```

Stili e temi

Altro widget è il **Table**, utilizzato per creare una struttura di righe e colonne con l'obiettivo di mostrare i dati i modo leggibile. Richiede una dimensione specifica delle colonne e che tutte le celle siano valorizzate.

Stili e temi

```
12    Widget build(BuildContext context) {
13      return MaterialApp(
14        home: Scaffold(
15          appBar: AppBar(
16            title: const Text('Table Example'),
17          ), // AppBar
18          body: Center(
19            child: Table(
20              border: TableBorder.all(),
21              children: [
22                TableRow(
23                  children: [
24                    TableCell(
25                      child: Container(
26                        padding: const EdgeInsets.all(8),
27                        color: Colors.blue,
28                        child: const Text(
29                          'Name',
30                          style: TextStyle(
31                            color: Colors.white,
32                            fontWeight: FontWeight.bold,
33                          ), // TextStyle
34                        ), // Text
35                      ), // Container
36                    ), // TableCell
```

Stili e temi

```
37                TableCell(
38                  child: Container(
39                    padding: const EdgeInsets.all(8),
40                    color: Colors.blue,
41                    child: const Text(
42                      'Age',
43                      style: TextStyle(
44                        color: Colors.white,
45                        fontWeight: FontWeight.bold,
46                      ), // TextStyle
47                    ), // Text
48                  ), // Container
49                ), // TableCell
50                TableCell(
51                  child: Container(
52                    padding: const EdgeInsets.all(8),
53                    color: Colors.blue,
54                    child: const Text(
55                      'City',
56                      style: TextStyle(
57                        color: Colors.white,
58                        fontWeight: FontWeight.bold,
59                      ), // TextStyle
60                    ), // Text
61                  ), // Container
62                ), // TableCell
```

Stili e temi

```
63                ],
64              ), // TableRow
65              TableRow(
66                children: [
67                  TableCell(
68                    child: Container(
69                      padding: const EdgeInsets.all(8),
70                      color: Colors.grey[300],
71                      child: const Text('John'),
72                    ), // Container
73                  ), // TableCell
74                  TableCell(
75                    child: Container(
76                      padding: const EdgeInsets.all(8),
77                      color: Colors.grey[300],
78                      child: const Text('25'),
79                    ), // Container
80                  ), // TableCell
81                  TableCell(
82                    child: Container(
83                      padding: const EdgeInsets.all(8),
84                      color: Colors.grey[300],
85                      child:const Text('New York'),
86                    ), // Container
87                  ), // TableCell
88                ],
89              ), // TableRow
```

Stili e temi

```
90                  TableRow(
91                    children: [
92                      TableCell(
93                        child: Container(
94                          padding: const EdgeInsets.all(8),
95                          color: Colors.grey[300],
96                          child: const Text('Jane'),
97                        ), // Container
98                      ), // TableCell
99                      TableCell(
100                       child: Container(
101                         padding: const EdgeInsets.all(8),
102                         color: Colors.grey[300],
103                         child: Text('30'),
104                       ), // Container
105                     ), // TableCell
106                     TableCell(
107                       child: Container(
108                         padding: const EdgeInsets.all(8),
109                         color: Colors.grey[300],
110                         child: const Text('London'),
111                       ), // Container
112                     ), // TableCell
113                   ],
114                 ), // TableRow
115               ],
116             ), // Table
117           ), // Center
118         ), // Scaffold
119       ); // MaterialApp
120     }
121   }
```

Stili e temi

In questo esempio, si ha un widget Table come corpo dello Scaffold. All'interno della tabella, si definisce la struttura della tabella utilizzando i widget TableRow e TableCell.

Stili e temi

La tabella ha un bordo definito con TableBorder.all(). Ogni riga è rappresentata da un widget TableRow e ogni cella all'interno di una riga è rappresentata da un widget TableCell.

Stili e temi

Le intestazioni della tabella vengono definite nella prima riga utilizzando i widget Container con colore di sfondo blu e testo bianco. Le righe successive contengono i dati con colore di sfondo grigio.

Stili e temi

È possibile personalizzare la struttura della tabella, il contenuto delle celle e lo stile in base alle proprie esigenze. Il widget Tabella offre un modo flessibile per visualizzare dati tabellari in un formato simile a una griglia.

ListWiev

ListView è tra i widget più importanti utilizzati frequentemente nelle applicazioni in Flutter. Secondo la documentazione, è un widget che mostra un elenco scrollabile di widget disposti in modo lineare (riga o colonna).

ListWiev

Per questo motivo necessita di un widget contenitore per determinare l'asse principale che definisce la direzione.

ListWiev

```
final List<String> entries = <String>['A', 'B', 'C'];
final List<int> colorCodes = <int>[600, 500, 100];

class MyApp extends StatelessWidget {
  const MyApp({super.key});


 @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home : Container(
        child : (
ListView.separated(
    padding: const EdgeInsets.all(20),
    itemCount: entries.length,
    itemBuilder: (BuildContext context, int index) {
      return Container(
        height: 50,
        color: Colors.amber[colorCodes[index]],
        child: Center(child:
        Text
        ('Entry ${entries[index]}',
         style: const TextStyle(
    color: Colors.white,
    fontSize: 16,
        ), // TextStyle
        ) // Text
        ), // Center
      ); // Container
    },
    separatorBuilder: (BuildContext context, int index) => const Divider(),
  ) // ListView.separated
        )
      ) // Container
    );       // MaterialApp
  }
}
```

Attività

Utilizzando gli stili e i widget di flutter simulare la rappresentazione l logo del browser chrome :

Introduzione alle Animazioni

Le animazioni aiutano a percepire l'applicazione come fluida e moderna. Molti widget di flutter, soprattutto i widget di material design, hanno numerose animazioni da poter utilizzare.

Introduzione alle Animazioni

In generale in flutter esistono due tipi di animazioni : Tween e Physics based animations.

Introduzione alle Animazioni

In Flutter, le animazioni tween sono utilizzate per creare transizioni fluide tra valori diversi per una durata specificata. Il termine "tween" sta per "in-between", a indicare che l'animazione interpola tra due valori per creare un effetto visivo di cambiamento continuo.

Introduzione alle Animazioni

Le animazioni tween sono comunemente utilizzate per animare proprietà come posizione, dimensione, opacità e colore dei widget nelle applicazioni Flutter.

Introduzione alle Animazioni

La classe Tween di Flutter fornisce un modo pratico per definire l'intervallo di valori per l'animazione e la classe AnimationController viene utilizzata per controllare la durata e la riproduzione dell'animazione.

Introduzione alle Animazioni

Un'animazione basata sulla fisica, invece, si basa sull'interazione con l'utente.
Un buon esempio è il fling: quanto più forte è lo scorrimento del dito su un lungo elenco scorrevole, più veloce sarà lo scorrimento.

Introduzione alle Animazioni

Un tween è dunque un oggetto con un valore iniziale e finale.

Ad esempio, un tween che cambia colore dal blu al rosa si servirà della animation library per determinare le sfumature intermedie per ogni fase dell'animazione.

Curve

La velocità delle animazioni si basano sulle curve. Una curva viene utilizzata per regolare il tasso di cambiamento di un'animazione nel tempo, consentendo all'animazione di accelerare o rallentare in punti specifici invece di muoversi a velocità costante.

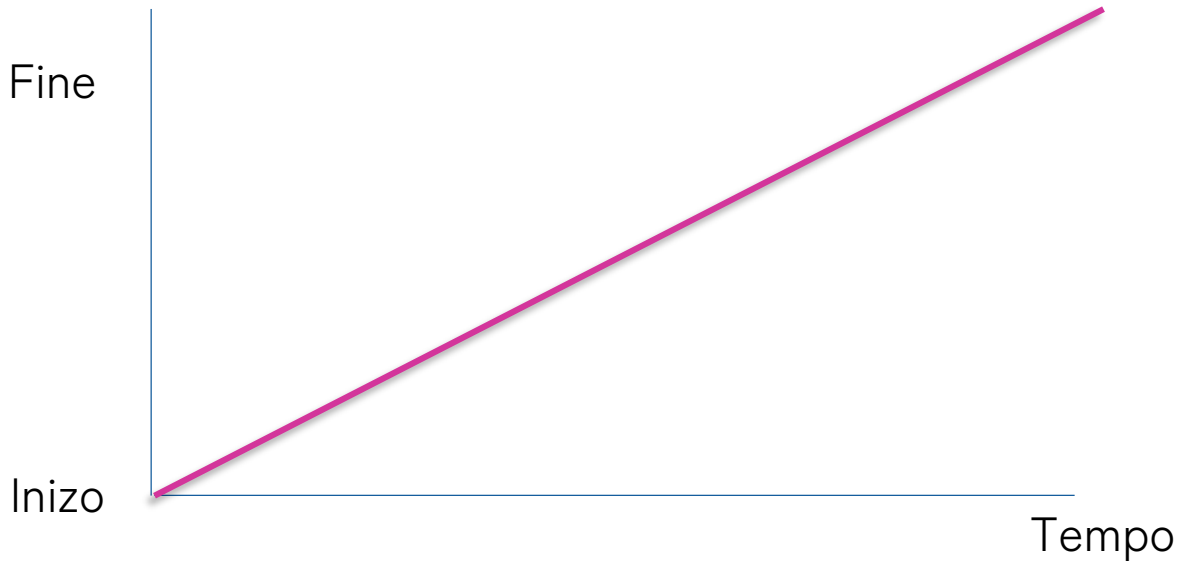
Curve

Flutter viene fornito con una serie di curve comuni e predefinite nella classe Curves. La curva curva predefinita è detta lineare, perché si muove a velocità costante.

La comprensione delle curve si ottiene confrontando una curva lineare, che è quella predefinita, con un'altra curva comune, chiamata cura ease in.

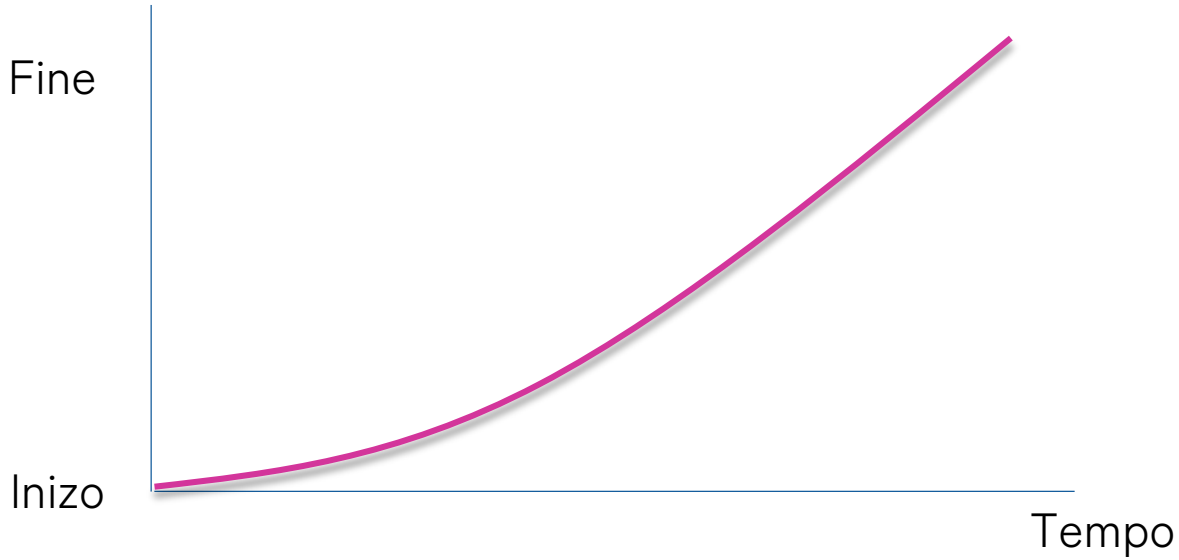
Curve

Curva lineare



Curve

Curva ease in : si avvia lentamente per poi procedere a velocità costante



Ticker

Nelle animazioni Flutter, un ticker è un meccanismo utilizzato per sincronizzare le animazioni con la frequenza di aggiornamento del dispositivo. I ticker sono gestiti dall'interfaccia TickerProvider, che consente ai widget di creare e controllare animazioni che vengono eseguite a una frequenza di fotogrammi costante. I ticker sono essenziali per garantire animazioni fluide ed efficienti nelle applicazioni Flutter.

AnimationController

Le animazioni sono gestite dall'AnimationController il quale è a conoscenza dell'oggetto Ticker che informa il controller di ogni fotogramma dell'animazione.

AnimationController

La classe AnimationController contiene metodi che avviano e interrompono le animazioni, resettare un'animazione, riprodurre un'animazione al contrario e ripeterla all'infinito. La classe ha anche dei getter che forniscono informazioni sull'animazione in corso.

AnimationController

I parametri necessari per creare un AnimationController sono un ticker e una durata di tempo.

AnimationController

Quando si crea un controller delle animazioni in Flutter, è necessario un parametro **vsync** che specifichi un TickerProvider.

AnimationController

Questo TickerProvider è responsabile di fornire un oggetto ticker che guida l'animazione segnalando quando un nuovo fotogramma è pronto per essere renderizzato. Il ticker aggiorna quindi il valore dell'animazione in base al tempo trascorso e notifica al controllore dell'animazione di ricostruire il widget con i valori aggiornati.

AnimatedWidget

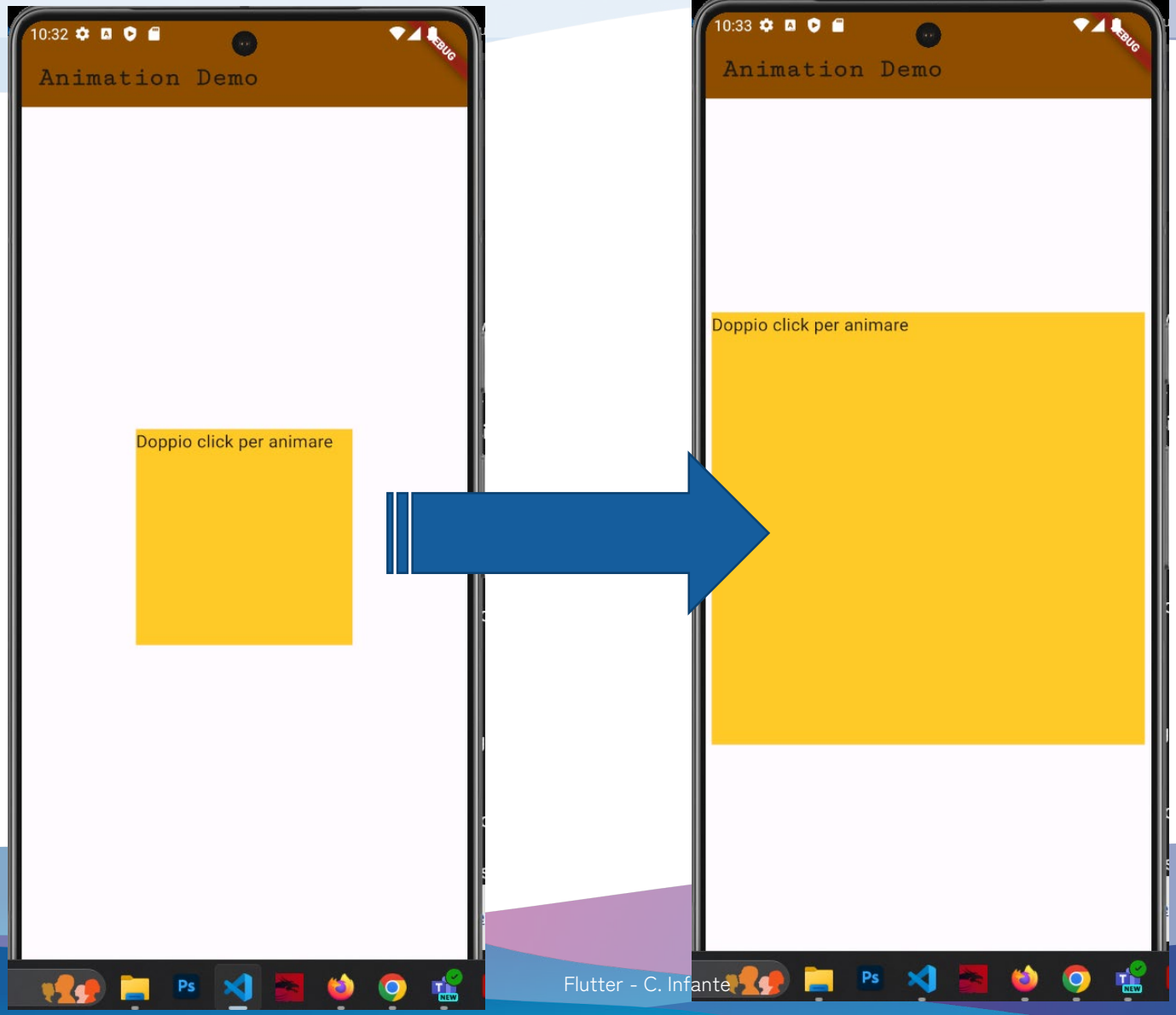
L'ultima cosa di cui ha bisogno un'animazione è il widget che si vuole animare.

Il widget che si vuole animare non è lo stesso widget che estende TickerStateProviderMixin, ma è piuttosto il figlio di quel widget.

Esempio

Si vuole realizzare una semplice animazione di questo tipo : un elemento centrale con del testo che, al doppio click raddoppia le proprie dimensioni.

Esempio


10:32
Animation Demo
Doppio click per animare
10:33
Animation Demo
Doppio click per animare

Esempio

Si parte dal MyApp widget, il widget Stateless che rappresenta la radice dell'applicazione in cui si definice il tema dell'intera applicazione.

Esempio

main.dart

lib > main.dart > _MyHomePageState

```dart
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Animation Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.amber.shade800),
        useMaterial3: true,
        textTheme: const TextTheme(
          titleLarge: TextStyle(
              fontSize: 24,
              fontWeight: FontWeight.bold,
              fontFamily: 'Courier New'), // TextStyle
          bodyMedium: TextStyle(fontSize: 16),
        ), // TextTheme
      ), // ThemeData
      home: const MyHomePage(title: 'Animation Demo'),
    ); // MaterialApp
  }
}
```

Esempio

Si Aggiorna il widget della homepage in modo tale che riprenda lo schema dei colori e il parametro del titolo settato in precedenza.

```
29
30  class MyHomePage extends StatefulWidget {
31    const MyHomePage({super.key, required this.title});
32
33    final String title;
34
35    @override
36    State<MyHomePage> createState() => _MyHomePageState();
37  }
38
```

Esempio

Si dovrà ora definire la struttura della pagina

```dart
class _MyHomePageState extends State<MyHomePage> {


  @override
  void initState() {
    super.initState();

  }


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
       title: Text(widget.title),
       backgroundColor: Theme.of(context).primaryColor,
      ), // AppBar
    ); // Scaffold
  }
}
```

```dart
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
        backgroundColor: Theme.of(context).primaryColor,
      ), // AppBar
      body: Center(
          child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Container(
            width: 200,
            height: 200,
            color: Colors.amber,
            child: const Text('Testo segnaposto'),
          ) // Container
        ], // <Widget>[]
      ) // Column
      ), // Center
    ); // Scaffold
  }
}
```

Esempio

Una volta determinata la struttura principale si andranno a gestire i cambiamenti delle animazioni grazie ai relativi controller, i quali saranno monitorati dal SingleTickerProviderStateMixin .

```
39  class _MyHomePageState extends State<MyHomePage> with SingleTickerProviderStateMixin {
40      late AnimationController _controller;
41    late Animation<double> _sizeAnimation;
42    @override
43    void initState() {
44      super.initState();
45      //Definizione dei cambiamenti dello stato e dunque dell'animazione
46    }
47
```

Esempio

Si aggiornerà anche la struttura, in modo che si determini l'azione e la funzione associata.

```
57      body: Center(
58          child: Column(
59        mainAxisAlignment: MainAxisAlignment.center,
60        children: <Widget>[
61          GestureDetector(
62              onDoubleTap: _animateSquare,
63            child: Container(
64                width: 200,
65                height: 200,
66                color: Colors.amber,
67                child: const Text('Doppio click per animare'),
68              ) // Container
69          ) // GestureDetector
70        ], // <Widget>[]
71      ) // Column
72      ), // Center
```

Esempio

Le proprietà relative la dimensione cambieranno grazie all'animazione, quindi saranno gestite in modo dinamico.

```
63          GestureDetector(
64              onDoubleTap: _animateSquare,
65              child: Container(
66                width: _sizeAnimation.value,
67                height: _sizeAnimation.value,
68                color: Colors.amber,
69                child: const Text('Doppio click per animare'),
70              )) // Container // GestureDetector
```

Esempio

Si definisca ora l'animation controller con le proprietà da gestire e la durata dell'animazione :

```
39  class _MyHomePageState extends State<MyHomePage> with SingleTickerProviderStateMixin {
40    late AnimationController _controller;
41    late Animation<double> _sizeAnimation;
42    @override
43    void initState() {
44      super.initState();
45      _controller = AnimationController(
46        vsync: this,
47        duration: const Duration(seconds: 1)
48        ); // AnimationController
49        _sizeAnimation = Tween<double>(begin: 200.0, end: 400.0).animate(_controller)
50        ..addListener(() {
51          setState(() {
52          });
53        });
54    }
55
```

Esempio

Si imposti anche il metodo per la gestione dello status del controller. In questo modo, ogni qualvolta si farà doppio click sull'elemento si avvierà l'animazione:

```
56    void _animateSquare() {
57      if (_controller.status == AnimationStatus.completed ) {
58        _controller.reverse();
59      } else {
60        _controller.forward();
61      }
62    }
```

Esempio

Se si desidera invece che l'elemento ritorni alla sua dimensione di partenza in automatico quando ha raggiunto la dimensione stabilita si andrà a definire un listener per il completamento dell'animazione

```
48        ); // AnimationController
49        _sizeAnimation = Tween<double>(begin: 200.0, end: 400.0).animate(_controller)
50        ..addListener(() {
51          setState(() {
52          });
53        })
54            ..addStatusListener((status) {
55            if (status == AnimationStatus.completed) {
56              _controller.reverse();
57            }
58          });
59      }
60
```

Esempio

Oltre alla dimensione è possibile gestire anche altre proprietà, come ad esempio il colore.

Esempio

Si definisce una nuova proprietà da controllare

```
39  class _MyHomePageState extends State<MyHomePage>
40      with SingleTickerProviderStateMixin {
41    late AnimationController _controller;
42    late Animation<double> _sizeAnimation;
43    late Animation<Color?> _colorAnimation;
44
```

Esempio

E il Tween associato con i valori iniziali e finali

```
55            ..addStatusListener((status) {
56              if (status == AnimationStatus.completed) {
57                _controller.reverse();
58              }
59            });
60      _colorAnimation =
61          ColorTween(begin: Colors.amber[200], end: Colors.amber[900])
62              .animate(_controller);
63    }
64
65    void _animateSquare() {
```

Esempio

Anche la proprietà color sarà gestita dunque facendo riferimento al controller

```
children: <Widget>[
  GestureDetector(
      onDoubleTap: _animateSquare,
      child: Container(
        width: _sizeAnimation.value,
        height: _sizeAnimation.value,
        color: _colorAnimation.value ?? Colors.amber,
        child: const Text('Doppio click per animare'),
      )) // Container // GestureDetector
], // <Widget>[]
), // Column // Center
// Scaffold
```

Esempio

Per eliminare l'animazione e per favorire le prestazioni dell'applicazione, ne interrompiamo l'esecuzione alla chiusura dell'applicazione o al cambio di pagina.

```
65    void _animateSquare() {
66      if (_controller.status == AnimationStatus.completed) {
67        _controller.reverse();
68      } else {
69        _controller.forward();
70      }
71    }
72    @override
73    void dispose() {
74      _controller.dispose();
75      super.dispose();
76    }
77
78    @override
79    Widget build(BuildContext context) {
```

Esempio

Per cambiare la linearità dell'animazione si può aggiungere la proprietà *curve.*

```
    AnimationController(vsync: this, duration: const Duration(seconds: 1));
_sizeAnimation =
    Tween<double>(begin: 200.0, end: 400.0).animate(
  CurvedAnimation(parent: _controller, curve: Curves.easeInOut),
)
```

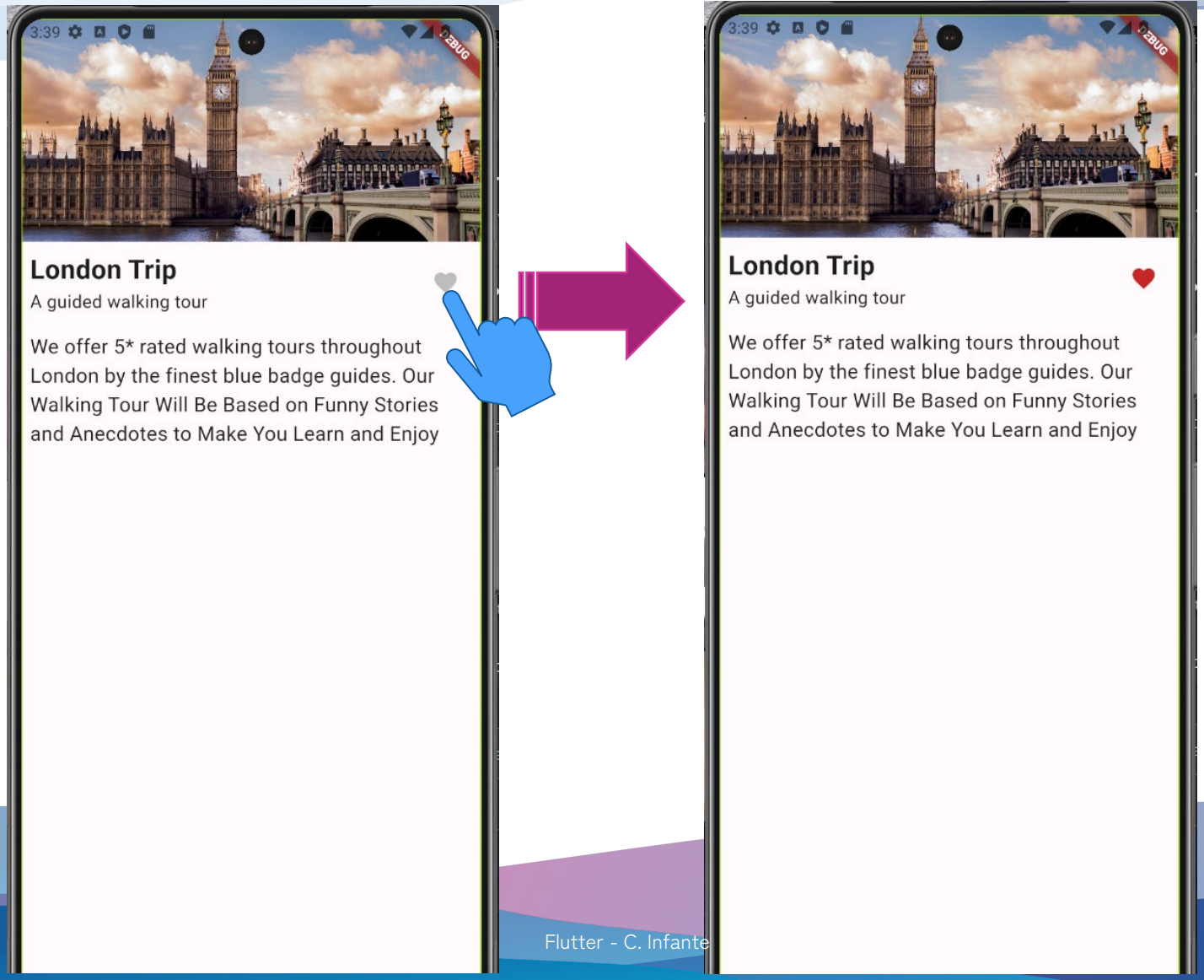
Domande e approfondimenti

Attività

Si strutturi una pagina come quella mostrata nell'esempio e si simuli sfruttando le animazioni, l'operazione di un salvataggio dell' elemento tra i preferiti.

Al click l'elemento aumenterà momentaneamente le proprie dimensioni, da grigio passerà a rosso e resterà tale. Se invece il colore di partenza è già rosso, al singolo click diventerà grigio.

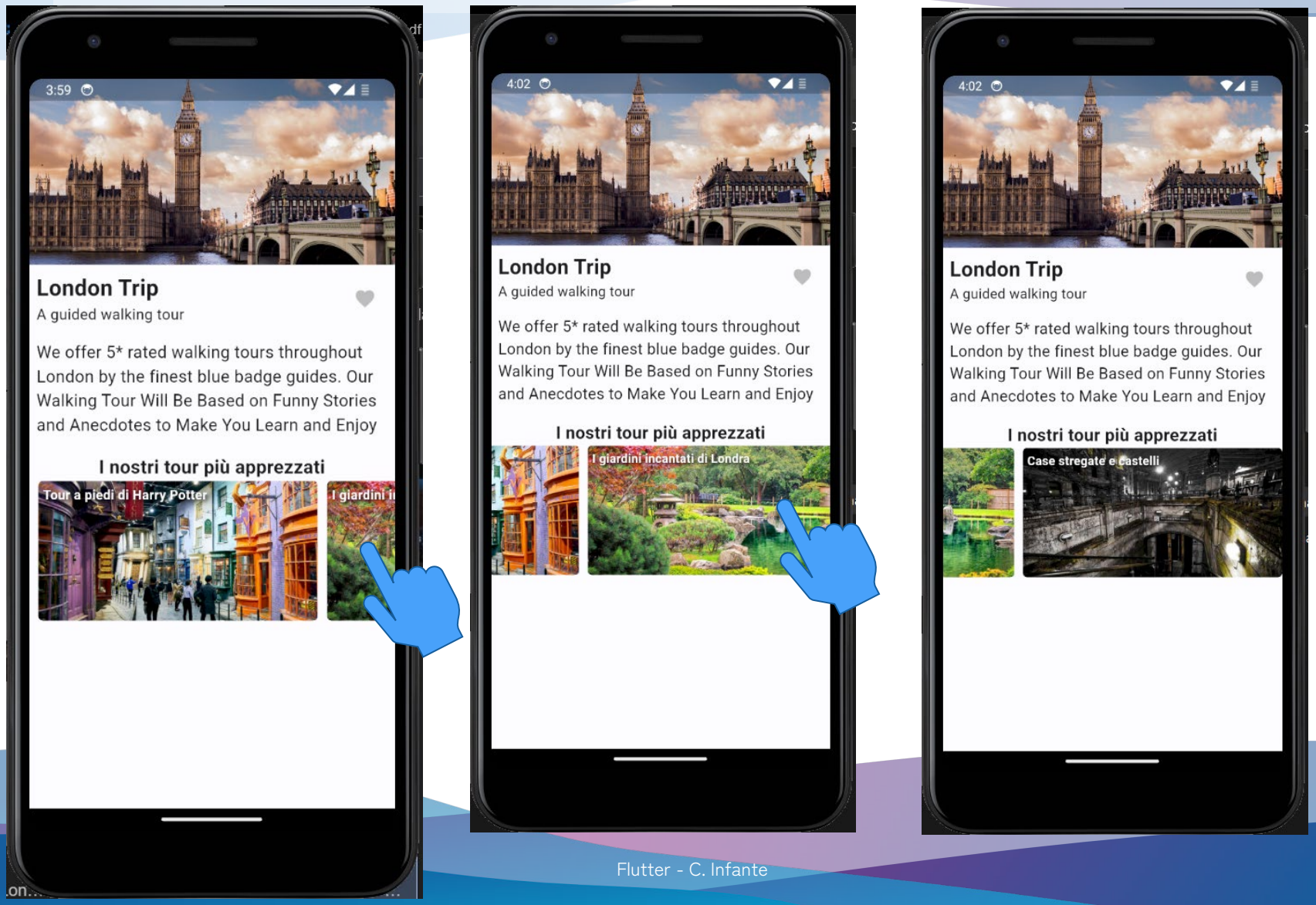
Attività


3:39
DEBUG
London Trip
A guided walking tour
We offer 5* rated walking tours throughout London by the finest blue badge guides. Our Walking Tour Will Be Based on Funny Stories and Anecdotes to Make You Learn and Enjoy
3:39
DEBUG
London Trip
A guided walking tour
We offer 5* rated walking tours throughout London by the finest blue badge guides. Our Walking Tour Will Be Based on Funny Stories and Anecdotes to Make You Learn and Enjoy

Attività

Definire inoltre una sezione che consenta di spostarsi orizzontalmente tra degli elementi stilizzati implementando delle animazioni.

Attività


3:59
London Trip
A guided walking tour
We offer 5* rated walking tours throughout London by the finest blue badge guides. Our Walking Tour Will Be Based on Funny Stories and Anecdotes to Make You Learn and Enjoy
I nostri tour più apprezzati
Tour a piedi di Harry Potter
4:02
London Trip
A guided walking tour
We offer 5* rated walking tours throughout London by the finest blue badge guides. Our Walking Tour Will Be Based on Funny Stories and Anecdotes to Make You Learn and Enjoy
I nostri tour più apprezzati
I giardini incantati di Londra
4:02
London Trip
A guided walking tour
We offer 5* rated walking tours throughout London by the finest blue badge guides. Our Walking Tour Will Be Based on Funny Stories and Anecdotes to Make You Learn and Enjoy
I nostri tour più apprezzati
Case stregate e castelli