

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

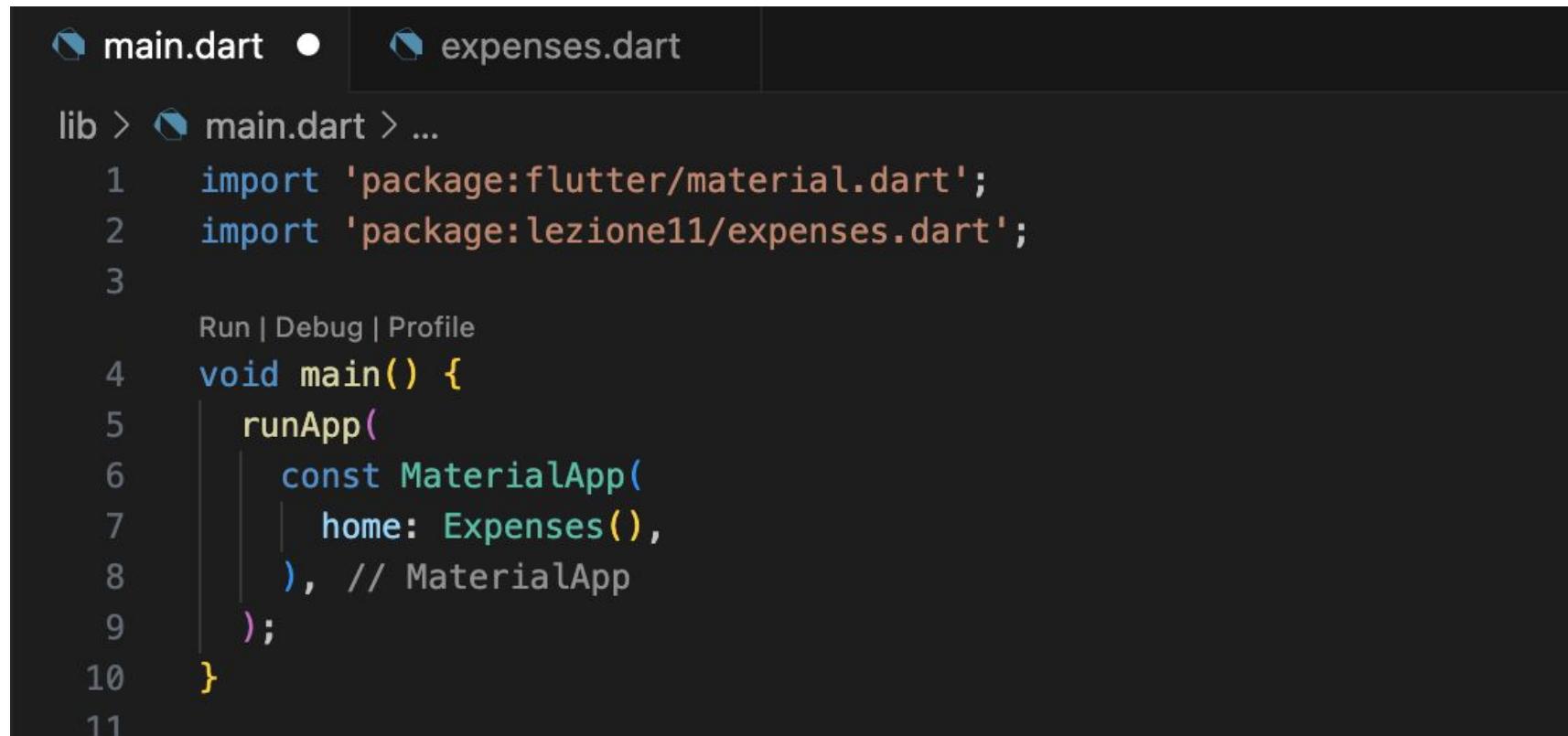
Lezione: Gestione liste e input

- Utilizzo delle ListView
- Utilizzo di input specifici
- Modali
- Date Picker
- Dialog
- Snackbar
- Switch tra temi



Progettiamo un
semplice tracker
per le spese
personalì

Cominciamo dal main.dart

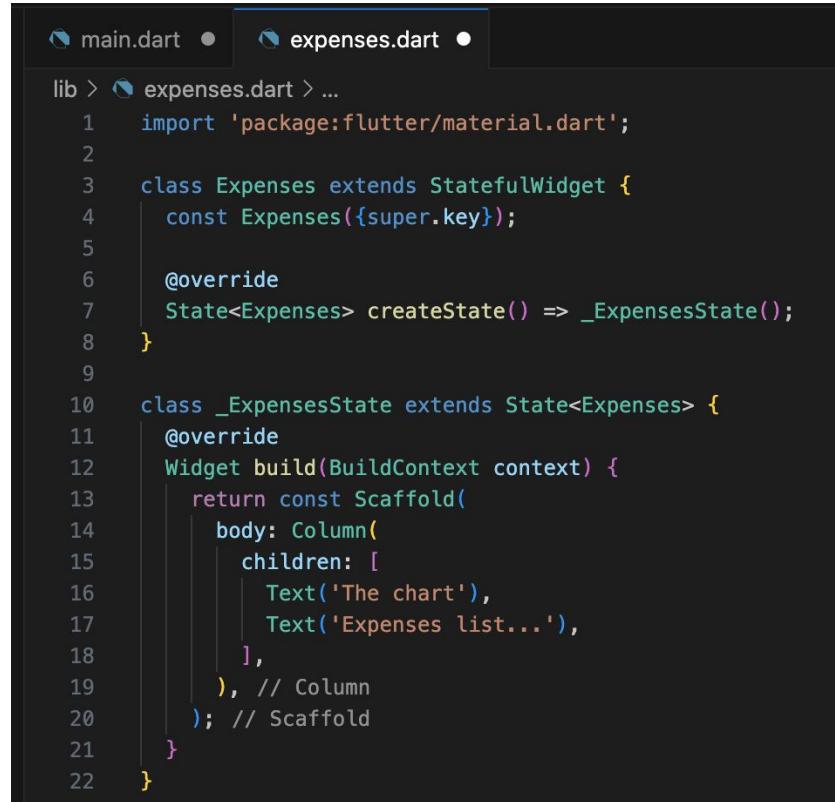


The screenshot shows a code editor interface with two tabs at the top: 'main.dart' (selected) and 'expenses.dart'. Below the tabs, a navigation bar indicates the file structure: 'lib > main.dart > ...'. The main content area displays the code for 'main.dart'.

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:lezione11/expenses.dart';
3
4 void main() {
5   runApp(
6     const MaterialApp(
7       home: Expenses(),
8     ), // MaterialApp
9   );
10 }
11
```

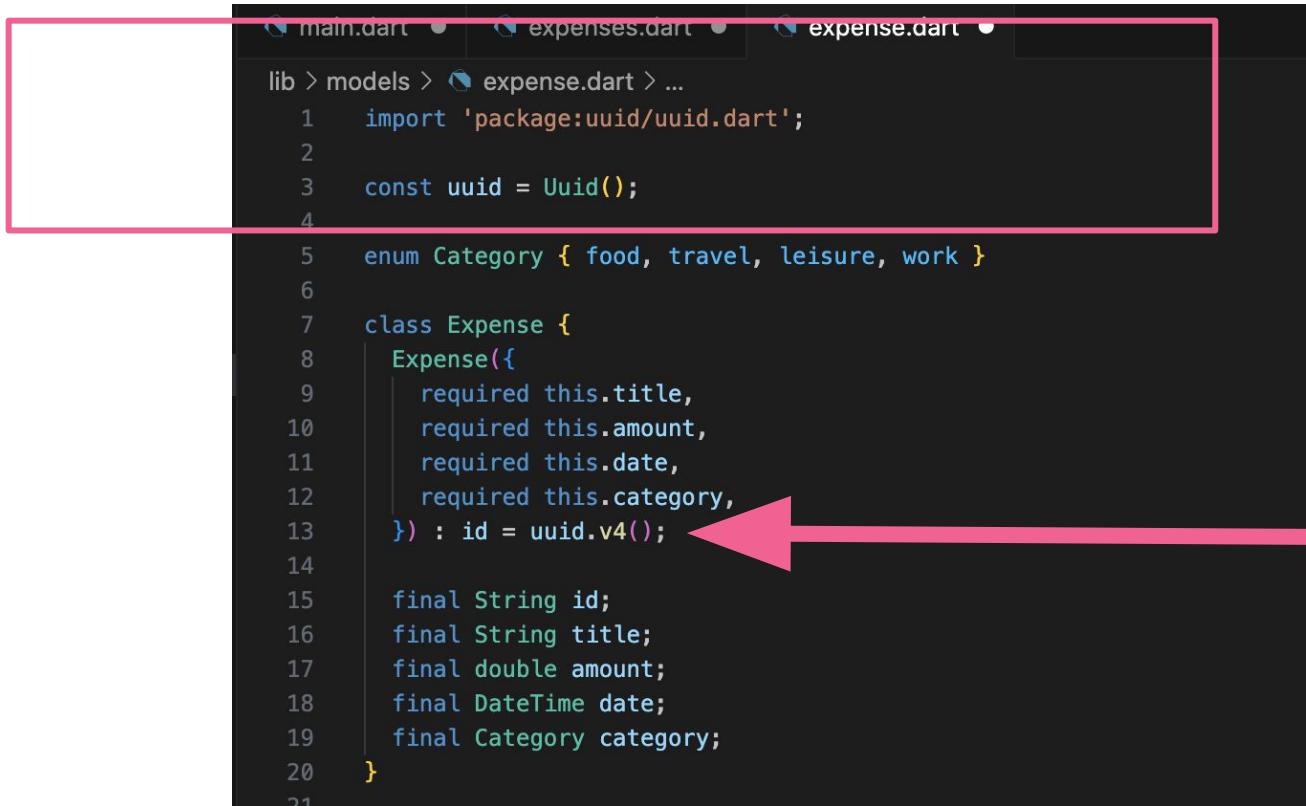
The code imports the necessary packages and defines the main function which runs the Expenses() widget as the home screen of a MaterialApp.

Creiamo Expenses, il Widget Stateful principale



```
lib > expenses.dart > ...
1   import 'package:flutter/material.dart';
2
3   class Expenses extends StatefulWidget {
4       const Expenses({super.key});
5
6       @override
7       State<Expenses> createState() => _ExpensesState();
8   }
9
10  class _ExpensesState extends State<Expenses> {
11      @override
12      Widget build(BuildContext context) {
13          return const Scaffold(
14              body: Column(
15                  children: [
16                      Text('The chart'),
17                      Text('Expenses list...'),
18                  ],
19              ), // Column
20          ); // Scaffold
21      }
22  }
```

uuid - <https://pub.dev/packages/uuid>



```
lib > models > expense.dart > ...
1   import 'package:uuid/uuid.dart';
2
3   const uuid = Uuid();
4
5   enum Category { food, travel, leisure, work }
6
7   class Expense {
8     Expense({
9       required this.title,
10      required this.amount,
11      required this.date,
12      required this.category,
13    }) : id = uuid.v4(); ←
14
15     final String id;
16     final String title;
17     final double amount;
18     final DateTime date;
19     final Category category;
20   }
21
```



Initializer Lists

```
main.dart • expenses.dart • expense.dart •  
lib > models > expense.dart > ...  
1 import 'package:uuid/uuid.dart';  
2  
3 const uuid = Uuid();  
4  
5 enum Category { food, travel, leisure, work }  
6  
7 class Expense {  
8     Expense({  
9         required this.title,  
10        required this.amount,  
11        required this.date,  
12        required this.category  
13    }) : id = uuid.v4();  
14  
15     final String id;  
16     final String title;  
17     final double amount;  
18     final DateTime date;  
19     final Category category;  
20 }  
21
```

In Dart gli Initializer Lists sono usati per inizializzare una determinata proprietà di una classe (come ad esempio un id) senza che questo sia passato al costruttore come argomento

enums

```
lib > models > expense.dart > ...
1   import 'package:uuid/uuid.dart';
2
3   const uuid = Uuid();
4
5   enum Category { viaggi, lavoro, hobby, cibo }
6
7   class Expense {
8     Expense({
9       required this.title,
10      required this.amount,
11      required this.date,
12      required this.category,
13    }) : id = uuid.v4();
14
15   final String id;
16   final String title;
17   final double amount;
18   final DateTime date;
19   final Category category;
20 }
```

Creiamo delle spese di default

```
class Expenses extends StatefulWidget {
  State<Expenses> createState() => _ExpensesState();
}

class _ExpensesState extends State<Expenses> {
  final List<Expense> registeredExpenses = [
    Expense(
      title: 'Talk di Carlo Lucera',
      amount: 0.0, // è gratis, dovete esserci!!!!
      date: DateTime.now(),
      category: Category.lavoro,
    ), // Expense
    Expense(
      title: 'Tour Guidato di Capracotta',
      amount: 29.99,
      date: DateTime.now(),
      category: Category.viaggi,
    ), // Expense
    Expense(
      title: 'Espresso al bar Unimol',
      amount: 1.00,
      date: DateTime.now(),
      category: Category.cibo,
    ), // Expense
  ];
}
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Column(
      children: [
        const Text('The chart'),
        Expanded(child: ExpensesList(listaspese: registeredExpenses)),
      ],
    ), // Column
  ); // Scaffold
}
}*
```

Gestiamo la lista delle spese in un nuovo widget

```
class Expenses extends StatefulWidget {
  State<Expenses> createState() => _ExpensesState();
}

class _ExpensesState extends State<Expenses> {
  final List<Expense> registeredExpenses = [
    Expense(
      title: 'Talk di Carlo Lucera',
      amount: 0.0, // è gratis, dovete esserci!!!!
      date: DateTime.now(),
      category: Category.lavoro,
    ), // Expense
    Expense(
      title: 'Tour Guidato di Capracotta',
      amount: 29.99,
      date: DateTime.now(),
      category: Category.viaggi,
    ), // Expense
    Expense(
      title: 'Espresso al bar Unimol',
      amount: 1.00,
      date: DateTime.now(),
      category: Category.cibo,
    ), // Expense
  ];
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Column(
      children: [
        const Text('The chart'),
        Expanded(child: ExpensesList(listaspese: registeredExpenses)),
      ],
    ), // Column
  ); // Scaffold
}
}
```

ListView

```
lib > expenses_list.dart > ...
1   import 'package:flutter/material.dart';
2
3   import 'package:lezione11/models/expense.dart';
4
5   class ExpensesList extends StatelessWidget {
6       const ExpensesList({
7           super.key,
8           required this.listaspese,
9       });
10
11   final List<Expense> listaspese;
12
13   @override
14   Widget build(BuildContext context) {
15       return ListView.builder(
16           itemCount: listaspese.length,
17           itemBuilder: (ctx, index) => Text(listaspese[index].title),
18       ); // ListView.builder
19   }
20 }
```

Gestiamo la lista delle spese in un nuovo widget

```
class Expenses extends StatefulWidget {
  State<Expenses> createState() => _ExpensesState();
}

class _ExpensesState extends State<Expenses> {
  final List<Expense> registeredExpenses = [
    Expense(
      title: 'Talk di Carlo Lucera',
      amount: 0.0, // è gratis, dovete esserci!!!!
      date: DateTime.now(),
      category: Category.lavoro,
    ), // Expense
    Expense(
      title: 'Tour Guidato di Capracotta',
      amount: 29.99,
      date: DateTime.now(),
      category: Category.viaggio,
    ), // Expense
    Expense(
      title: 'Espresso al bar Unimol',
      amount: 1.00,
      date: DateTime.now(),
      category: Category.cibo,
    ), // Expense
  ];
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    body: Column(
      children: [
        const Text('The chart'),
        Expanded(child: ExpensesList(listaspese: registeredExpenses)),
      ],
    ), // Column
  ); // Scaffold
}
```

Uso Expanded perché in flutter non posso mettere direttamente una ListView in una Colum

ListView

Cos'è una ListView: Un widget che mostra un elenco scrollabile di elementi in un'app Flutter.

Caratteristiche Principali

- Visualizzazione efficiente di grandi insiemi di dati;
- Supporta la costruzione di layout verticali e orizzontali;
- Può contenere una varietà di widget per rappresentare gli elementi della lista;

Utilizzo

- Mostrare elenchi di elementi come articoli in un negozio, messaggi in una chat o contatti in un'app di contatti;
- Gestire in maniera semplice gli eventi degli elementi come il tap o lo swipe;

Personalizzazione

- Personalizzabile tramite vari parametri come itemBuilder, separatorBuilder e scrollDirection;
- Possibilità di personalizzare l'aspetto degli elementi tramite widget personalizzati gestendo comunque la memoria in maniera ottimizzata.

ListView vs Column

Quando usare una ListView

- **Grande quantità di dati:** Se hai un grande insieme di dati da visualizzare, come una lista di elementi dinamica caricati da un database o da una chiamata API, una ListView è più efficiente in quanto carica e visualizza solo gli elementi attualmente visibili sulla schermata, riducendo la memoria e il carico del processore;
- **Scorrevolezza:** ListView offre nativamente la capacità di scorrere in modo fluido attraverso gli elementi della lista. Gli utenti possono scorrere su e giù per visualizzare tutti gli elementi senza doverli disporre manualmente;
- **Gestione degli eventi di scorrimento:** ListView gestisce automaticamente gli eventi di scorrimento, consentendo agli utenti di interagire con la lista in modo intuitivo.

Quando usare una Column:

- Quantità limitata di dati: Se hai solo un piccolo numero di elementi da visualizzare e la lista non è destinata a cambiare dinamicamente, una Column può essere più semplice ed efficace;
- Disposizione verticale statica: Se gli elementi nella tua UI devono essere disposti staticamente uno sopra l'altro e non richiedono uno scorrimento, una Column può essere la scelta migliore.

Risultato fino ad ora





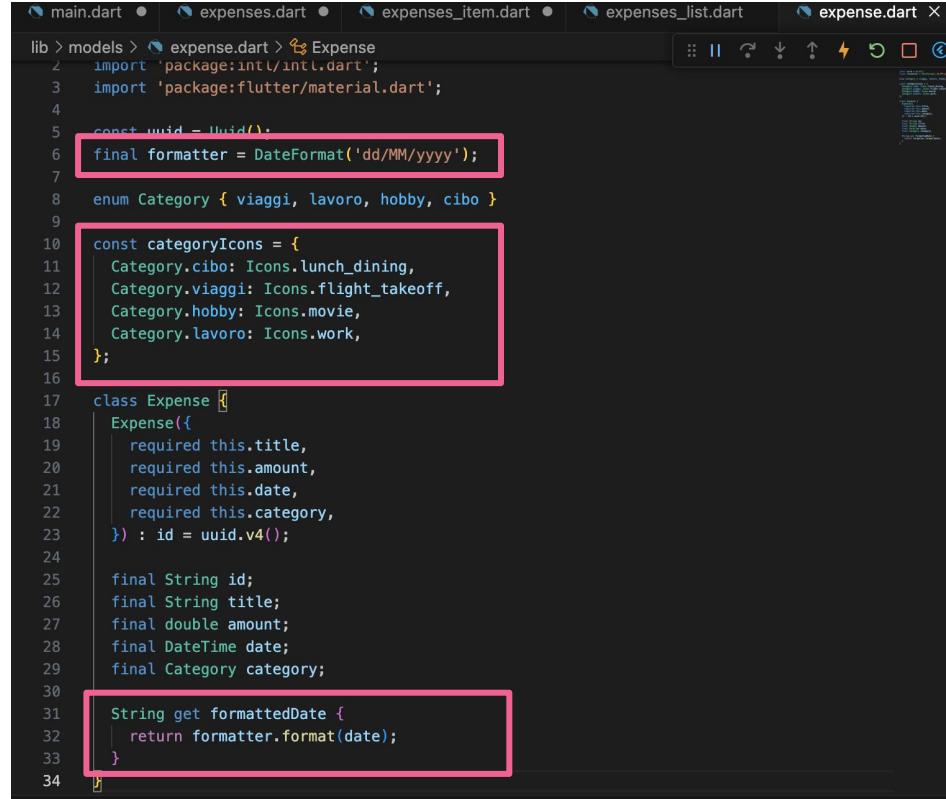
Personalizziamo gli
elementi della
nostra lista

intl - <https://pub.dev/packages/intl>

Per avere un supporto nella gestione delle date usiamo il Package **intl**.

Il Package **intl** è una libreria che fornisce supporto per la localizzazione e la formattazione internazionale nelle App. Con **intl**, è possibile formattare numeri, valori, date e testi in base alle preferenze locali dell'utente. Inoltre, offre strumenti per gestire la traduzione del testo e l'adattamento dell'interfaccia utente in diverse lingue e regioni. **intl** semplifica e supporta il processo di creazione di App Flutter che possono essere utilizzate in tutto il mondo, garantendo che l'esperienza dell'utente sia coerente e adattata alle sue preferenze linguistiche e culturali.

Integro il data model con icone e date



```
lib > models > expense.dart > Expense
1  import 'package:intl/intl.dart';
2  import 'package:flutter/material.dart';
3
4  const uid =Uuid();
5
6  final formatter = DateFormat('dd/MM/yyyy');
7
8  enum Category { viaggi, lavoro, hobby, cibo }
9
10 const categoryIcons = {
11   Category.cibo: Icons.lunch_dining,
12   Category.viaggi: Icons.flight_takeoff,
13   Category.hobby: Icons.movie,
14   Category.lavoro: Icons.work,
15 };
16
17 class Expense {
18   Expense({
19     required this.title,
20     required this.amount,
21     required this.date,
22     required this.category,
23   }) : id = uid.v4();
24
25   final String id;
26   final String title;
27   final double amount;
28   final DateTime date;
29   final Category category;
30
31   String get formattedDate {
32     return formatter.format(date);
33   }
34 }
```

Creo un nuovo Widget per un singolo elemento della lista

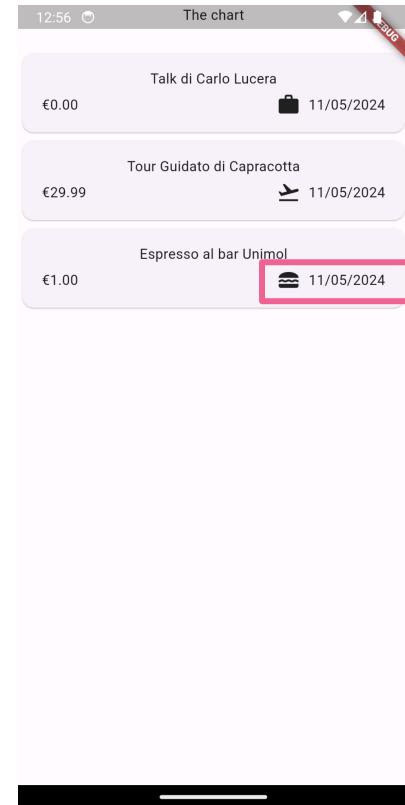
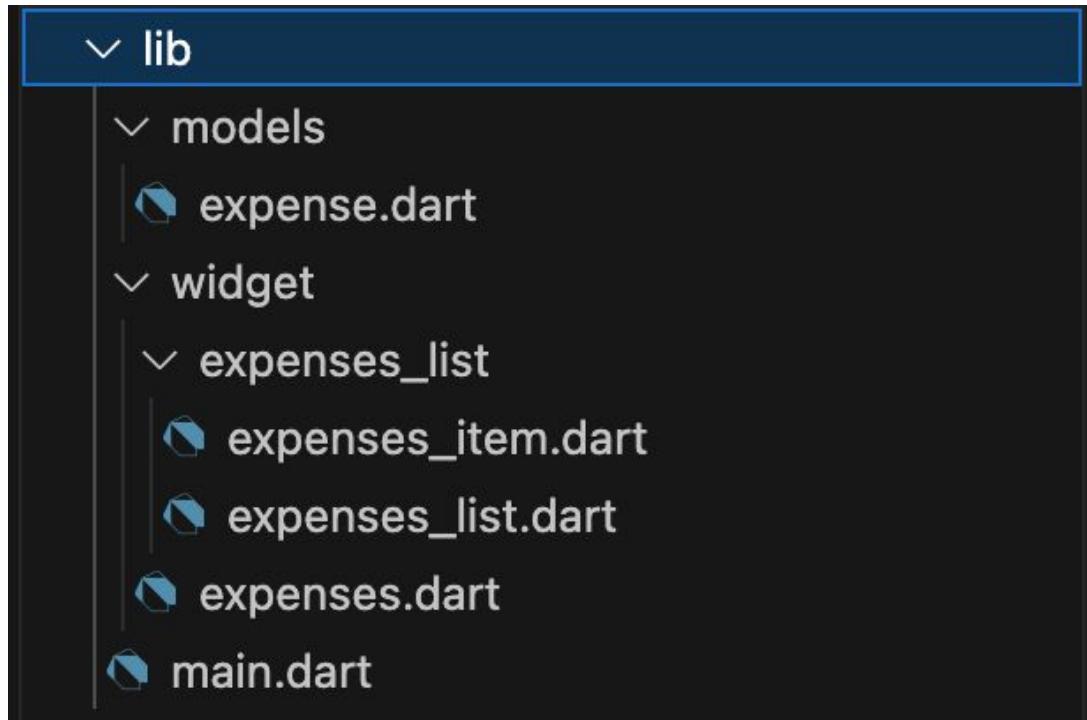


```
lib > widget > expenses_list > expenses_item.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:lezione11/models/expense.dart';
3
4 class ExpenseItem extends StatelessWidget {
5   const ExpenseItem(this.spesa, {super.key});
6
7   final Expense spesa;
8
9   @override
10  Widget build(BuildContext context) {
11    return Card(
12      child: Padding(
13        padding: const EdgeInsets.symmetric(
14          horizontal: 20,
15          vertical: 16,
16        ), // EdgeInsets.symmetric
17        child: Column(
18          children: [
19            Text(spesa.title),
20            const SizedBox(height: 4),
21            Row(
22              children: [
23                Text(
24                  '\u20ac${spesa.amount.toStringAsFixed(2)}', // fisso la dimensione a 2 decimali // Text
25                  const Spacer(),
26                  Row(
27                    children: [
28                      Icon(categoryIcons[spesa.category]),
29                      const SizedBox(width: 8),
30                      Text(spesa.formattedDate),
31                    ],
32                  ), // Row
33                ],
34              ), // Row
35            ],
36          ),
37        ), // Column
38      ), // Padding
39    ); // Card
40 }
```

Integro il Widget in quello della Lista

```
lib > widget > expenses_list > expenses_list.dart > ...
1  import 'package:flutter/material.dart';
2  import 'package:lezione11/widget/expenses_list/expenses_item.dart';
3  import 'package:lezione11/models/expense.dart';
4
5  class ExpensesList extends StatelessWidget {
6      const ExpensesList({
7          super.key,
8          required this.listaspese,
9      });
10
11     final List<Expense> listaspese;
12
13     @override
14     Widget build(BuildContext context) {
15         return ListView.builder(
16             itemCount: listaspese.length,
17             itemBuilder: (ctx, index) => ExpenseItem(listaspese[index]),
18         ); // ListView.builder
19     }
20 }
```

Riorganizzazione dei widget e risultato





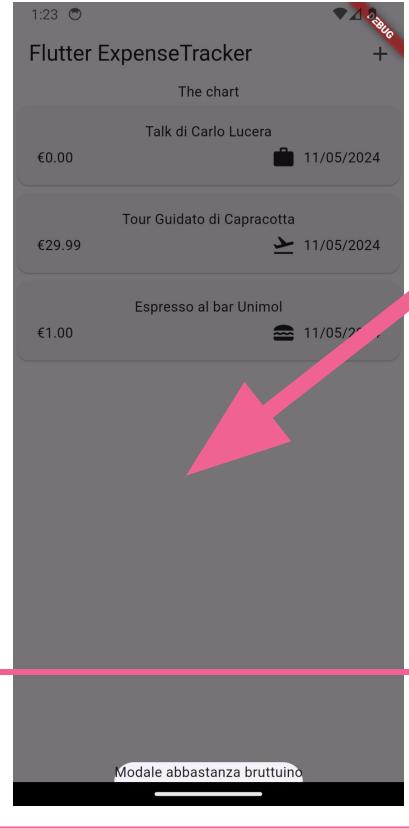
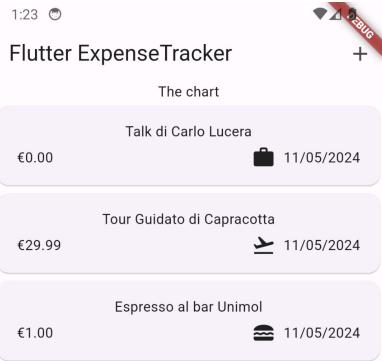
Aggiunta di un nuovo elemento

AppBar all'interno del widget Expenses

```
void _openAddExpenseOverlay() {  
    showModalBottomSheet(  
        context: context,  
        builder: (ctx) => const Text('Modale abbastanza bruttuino'),  
    );  
}  
  
@override  
Widget build(BuildContext context) {  
    return Scaffold(  
        appBar: AppBar(  
            title: const Text('Flutter ExpenseTracker'),  
            actions: [  
                IconButton(  
                    onPressed: _openAddExpenseOverlay,  
                    icon: const Icon(Icons.add),  
                ), // IconButton  
            ],  
        ), // AppBar
```



Risultato



Toccare nella zona grigia per chiudere il modale e ritornare alla schermata precedente

Creo il widget da usare nel modale: NewExpense

```
11
12 class _NewExpenseState extends State<NewExpense> {
13   var _enteredTitle = '';
14
15   void _saveTitleInput(String inputValue) {
16     _enteredTitle = inputValue;
17   }
18
19   @override
20   Widget build(BuildContext context) {
21     return Padding(
22       padding: const EdgeInsets.all(16),
23       child: Column(
24         children: [
25           TextField(
26             onChanged: _saveTitleInput,
27             maxLength: 50,
28             decoration: const InputDecoration(
29               label: Text('Titolo'),
30             ), // InputDecoration
31             ), // TextField
32           Row(
33             children: [
34               ElevatedButton(
35                 onPressed: () {
36                   print(_enteredTitle);
37                 },
38                 child: const Text('Aggiungi la spesa'),
39               ), // ElevatedButton
40             ],
41           ), // Row
42           ],
43         ), // Column
44       ); // Padding
45     }
46 }
```

Il widget è Stateful, perché sarà soggetto a cambiamenti, ma concentriamoci su alcune funzionalità che non abbiamo mai visto...

NewExpense

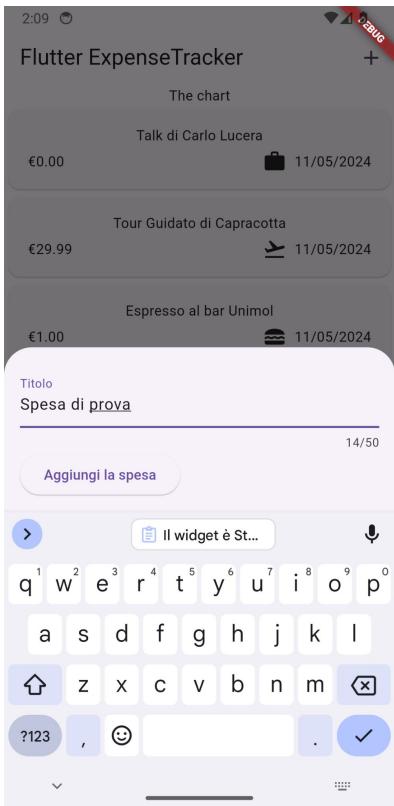
Modifico anche Expenses, per fare in modo che crei un'istanza di NewExpense all'apertura del modale.

```
class _NewExpenseState extends State<NewExpense> {
  var _enteredTitle = '';

  void _saveTitleInput(String inputValue) {
    _enteredTitle = inputValue;
  }

  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: const EdgeInsets.all(16),
      child: Column(
        children: [
          TextField(
            onChanged: _saveTitleInput,
            maxLength: 50,
            decoration: const InputDecoration(
              label: Text('Titolo'),
            ), // InputDecoration
          ), // TextField
          Row(
            children: [
              ElevatedButton(
                onPressed: () {
                  print(_enteredTitle);
                },
                child: const Text('Aggiungi la spesa'),
              ), // ElevatedButton
            ],
          ),
        ],
      ),
    );
  }
}
```

Il risultato è questo



```
D/EGL_emulation( 4779): app_time_stats: avg=501.00ms min=477.92ms max=523.63ms count=3
D/EGL_emulation( 4779): app_time_stats: avg=498.10ms min=496.62ms max=500.67ms count=3
D/EGL_emulation( 4779): app_time_stats: avg=500.10ms min=491.68ms max=505.79ms count=3
D/EGL_emulation( 4779): app_time_stats: avg=499.76ms min=499.35ms max=500.17ms count=2
D/EGL_emulation( 4779): app_time_stats: avg=499.37ms min=496.98ms max=501.11ms count=3
D/EGL_emulation( 4779): app_time_stats: avg=501.62ms min=500.02ms max=503.22ms count=2
I/flutter ( 4779): Spesa di prova
D/EGL_emulation( 4779): app_time_stats: avg=50.06ms min=6.96ms max=495.64ms count=20
D/EGL_emulation( 4779): app_time_stats: avg=71.03ms min=6.34ms max=508.18ms count=21
D/EGL_emulation( 4779): app_time_stats: avg=500.41ms min=499.91ms max=500.91ms count=2
D/EGL_emulation( 4779): app_time_stats: avg=500.22ms min=497.51ms max=502.92ms count=2
```

Ma possiamo fare di meglio: TextEditingController

```
class _NewExpenseState extends State<NewExpense> {  
    final _titleController = TextEditingController();  
    @override  
    void dispose() {  
        // serve per i memory leak  
        _titleController.dispose();  
        super.dispose();  
    }  
  
    @override  
    Widget build(BuildContext context) {  
        return Padding(  
            padding: const EdgeInsets.all(16),  
            child: Column(  
                children: [  
                    TextField(  
                        controller: _titleController,  
                        maxLength: 50,  
                        decoration: const InputDecoration(  
                            label: Text('Titolo'),  
                        ), // InputDecoration  
                    ), // TextField  
                    Row(  
                        children: [  
                            ElevatedButton(  
                                onPressed: () {  
                                    print(_titleController.text);  
                                },  
                                child: const Text('Aggiungi la spesa'),  
                            ),  
                        ],  
                    ),  
                ],  
            ),  
        );  
    }  
}
```

Il risultato è lo stesso, ma evitiamo gestioni manuali e abbiamo un uso ottimizzato della memoria

Esercizio

Partendo dal file esercizio1.zip modificate new_expense.dart per permettere l'aggiunta di un nuovo input (numerico) per l'inserimento degli importi in € e implementate un bottone “Cancel”

Aggiungiamo una data come input al nostro modale

```
class _NewExpenseState extends State<NewExpense> {
    final _titleController = TextEditingController();
    final _amountController = TextEditingController();
    DateTime? _selectedDate;

    void _presentDatePicker() async {
        final now = DateTime.now();
        final firstDate = DateTime(now.year - 1, now.month, now.day);
        final pickedDate = await showDatePicker(
            context: context,
            initialDate: now,
            firstDate: firstDate,
            lastDate: now,
        );
        setState(() {
            _selectedDate = pickedDate;
        });
    }
}
```

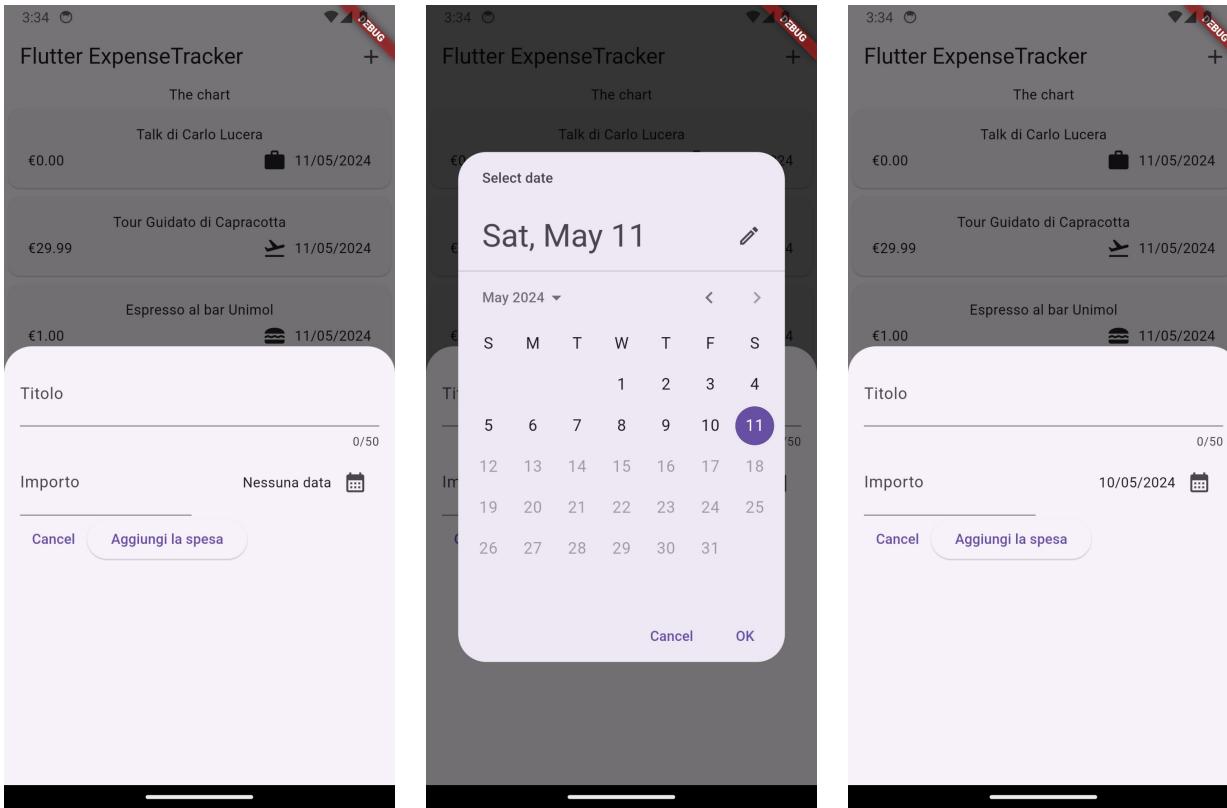


Aggiungiamo una data come input al nostro modale

```
const SizedBox(width: 16),  
Expanded(  
    child: Row(  
        mainAxisAlignment: MainAxisAlignment.end,  
        crossAxisAlignment: CrossAxisAlignment.center,  
        children: [  
            Text(  
                _selectedDate == null  
                    ? 'Nessuna data selezionata'  
                    : formatter.format(_selectedDate!),  
            ), // Text  
            IconButton(  
                onPressed: _presentDatePicker,  
                icon: const Icon(  
                    Icons.calendar_month,  
                ), // Icon  
            ), // IconButton
```



Il risultato



La parte di formattazione del resto del codice relativo a questa schermata non sarà spiegata in questa lezione poiché contiene elementi già mostrati nelle precedenti lezioni, è possibile comunque consultare l'intero codice allegato alla lezione odierna.

L'ultimo input: le categorie

```
    children: [
      DropdownButton(
        value: _selectedCategory,
        items: Category.values
          .map(
            (category) => DropdownMenuItem(
              value: category,
              child: Text(
                category.name.toUpperCase(),
              ), // Text
            ), // DropdownMenuItem
          )
          .toList(),
        onChanged: (value) {
          if (value == null) {
            return;
          }
          setState(() {
            _selectedCategory = value;
          });
        },
      ), // DropdownButton
      const Spacer(),
    ],
  );
}
```

items vuole un elemento di tipo DropdownMenuItem mentre Category è di tipo enum, per questo dobbiamo “convertirli” con .map

L'ultimo input: le categorie

```
class _NewExpenseState extends State<NewExpense> {  
    final _titleController = TextEditingController();  
    final _amountController = TextEditingController();  
    DateTime? _selectedDate;  
    Category _selectedCategory = Category.hobby;  
  
    void _presentDatePicker() async {  
        final now = DateTime.now();  
        final firstDate = DateTime(now.year - 1, now.month, no  
        final pickedDate = await showDatePicker(  
            context: context,  
            initialDate: now,  
            firstDate: firstDate,  
            lastDate: DateTime.now(),  
            builder: (BuildContext context) {  
                return Theme(...);  
            },  
        );  
        if (pickedDate != null && pickedDate != _selectedDate) {  
            setState(() {  
                _selectedDate = pickedDate;  
            });  
        }  
    }  
}
```

Inizializzo
_selectedCategory
per una questione
estetica e per evitare
di vedere “null” alla
categoria al
momento
dell’apertura del
modale

Salvataggio di un nuovo elemento

Cominciamo a definire un metodo per il salvataggio di un nuovo elemento alla pressione del bottone “Aggiungi la spesa”

```
        },
        child: const Text('Cancel'),
    ), // TextButton
ElevatedButton(
    onPressed: _submitExpenseData,
    child: const Text('Aggiungi la spesa'),
), // ElevatedButton
],
),
), // Row
],
),
), // Column
```

Valido i valori di Titolo e Importo

```
void _submitExpenseData() {
    final enteredAmount = double.tryParse(_amountController
        .text); // tryParse('prova') => null, tryParse('1.12') => 1.12
    final amountIsValid = enteredAmount == null || enteredAmount <= 0;
    if (_titleController.text.trim().isEmpty ||
        amountIsValid ||
        _selectedDate == null) {
        showDialog(
            ...
        );
    }
}
```

Definisco il messaggio di errore

```
showDialog(  
    context: context,  
    builder: (ctx) => AlertDialog(  
        title: const Text('Input non valido'),  
        content: const Text(  
            'Verifica che il titolo e l\'importo siano corretti e che tu abbia selezionato una data valida.'), // Text  
        actions: [  
            TextButton(  
                onPressed: () {  
                    Navigator.pop(ctx);  
                },  
                child: const Text('Capito!'),  
            ), // TextButton  
        ],  
    ), // AlertDialog  
);  
return;  
}
```

Devo aggiornare la lista delle spese che si trova nel widget Expenses, come faccio?

```
class NewExpense extends StatefulWidget {
  const NewExpense({super.key, required this.onAddExpense});

  final void Function(Expense expense) onAddExpense;

  @override
  State<NewExpense> createState() => _NewExpenseState();
}

class _NewExpenseState extends State<NewExpense> {
  final _titleController = TextEditingController();
  final _amountController = TextEditingController();
  DateTime? _selectedDate;
  Category _selectedCategory = Category.hobby;

  void _presentDatePicker() async {
    final DateTime? pickedDate = await showDatePicker(
      context: context,
      initialDate: DateTime.now(),
      firstDate: DateTime(2022),
      lastDate: DateTime(2025));
    if (pickedDate != null && pickedDate != _selectedDate)
      setState(() {
        _selectedDate = pickedDate;
      });
  }

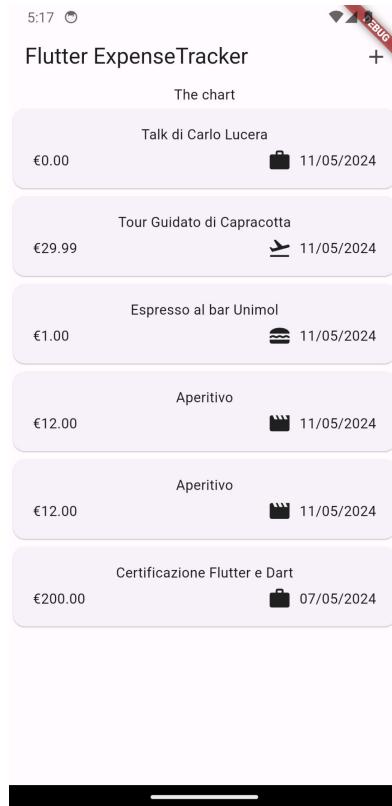
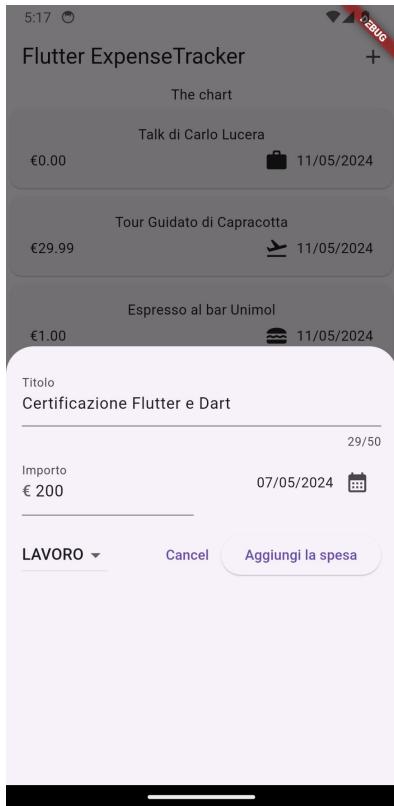
  void _submitExpenseData() {
    final enteredAmount = double.tryParse(_amountController.text);
    final amountIsInvalid = enteredAmount == null || enteredAmount <= 0;
    if (_titleController.text.trim().isEmpty ||
        amountIsInvalid ||
        _selectedDate == null) {
      return;
    }
    widget.onAddExpense(
      Expense(
        title: _titleController.text,
        amount: enteredAmount,
        date: _selectedDate!,
        category: _selectedCategory,
      ),
    );
  }
}
```

Devo
adeguare
anche
Expenses

Devo aggiornare la lista delle spese che si trova nel widget Expenses, come faccio?

```
2     ), // Expense
3 ];
4 void _addExpense(Expense expense) {
5     setState(() {
6         registeredExpenses.add(expense);
7     });
8 }
9
10 void _openAddExpenseOverlay() {
11     showModalBottomSheet(
12         context: context,
13         builder: (ctx) => NewExpense(onAddExpense: _addExpense),
14     );
15 }
```

Funziona, ma cosa manca?



La schermata non si chiude in automatico dopo l'aggiunta e un modale così basso mi crea problemi con la testiera perché si sovrappone. Nell'attesa di trovare una soluzione...

Esercizio

Partendo dal file esercizio2.zip aggiungete la Geolocalizzazione alle spese. Ogni spesa potrà contenere delle coordinate geografiche (opzionali).

Chiudere il modale dopo il salvataggio

```
widget.onAddExpense(  
    Expense(  
        title: _titleController.text,  
        amount: enteredAmount,  
        date: _selectedDate!,  
        category: _selectedCategory,  
    ),  
);  
Navigator.pop(context);  
}
```



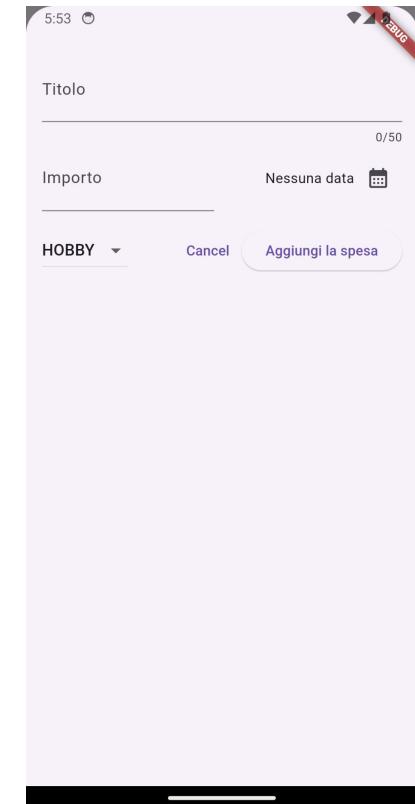
Portare il modale a tutto schermo

```
lib > widget > expenses.dart > _ExpensesState
13   class _ExpensesState extends State<Expenses> {
14     final List<Expense> registeredExpenses = [
15       Expense(
16         amount: 29.55,
17         date: DateTime.now(),
18         category: Category.viaggi,
19       ), // Expense
20       Expense(
21         title: 'Espresso al bar Unimol',
22         amount: 1.00,
23         date: DateTime.now(),
24         category: Category.cibo,
25       ), // Expense
26     ];
27     void _addExpense(Expense expense) {
28       setState(() {
29         registeredExpenses.add(expense);
30       });
31     }
32
33
34     void _openAddExpenseOverlay() {
35       showModalBottomSheet(
36         isScrollControlled: true,
37         context: context,
38         builder: (ctx) => NewExpense(onAddExpense: _addExpense),
39       );
40     }
41   }
```



Portare il modale a tutto schermo

```
lib > widget > new_expense.dart > _NewExpenseState > build
15   class _NewExpenseState extends State<NewExpense> {
35     void _submitExpenseData() {
65       date: _selectedDate!,
66       category: _selectedCategory,
67     ),
68   );
69   Navigator.pop(context);
70 }
71
72 @override
73 void dispose() {
74   // serve per i memory leak
75   _titleController.dispose();
76   _amountController.dispose();
77   super.dispose();
78 }
79
80 @override
81 Widget build(BuildContext context) {
82   return Padding(
83     padding: const EdgeInsets.fromLTRB(16, 52, 16, 16),
84     child: Column(
85       children: [
86         TextField( // TextField ...
87         Row( // Row ...
88           SizedBox(height: 16)
```





E se volessi
eliminare una
spesa?

Uso il widget Dismissible all'interno della ListView

```
expenses_list.dart ●
lib > widget > expenses_list > expenses_list.dart > ...
5   class ExpensesList extends StatelessWidget {
10
11     final List<Expense> listaspese;
12
13     @override
14     Widget build(BuildContext context) {
15       return ListView.builder(
16         itemCount: listaspese.length,
17         itemBuilder: (ctx, index) => Dismissible(
18           key: ValueKey(listaspese[index]),
19           child: ExpenseItem(
20             listaspese[index],
21           ), // ExpenseItem
22           ), // Dismissible
23         ); // ListView.builder
24     }
25
26
```



Per la prima volta valorizziamo un campo key: ci serve per capire quale elemento dobbiamo rimuovere

Uso il widget Dismissible all'interno della ListView

8:17

Flutter ExpenseTracker

The chart

Talk di Carlo Lucera
€0.00

11/05/2024

Tour Guidato di Capracotta
€29.99

11/05/2024

Effettuo la
gesture dello
slide a sinistra
e rimuovo un
elemento

8:17

Flutter ExpenseTracker

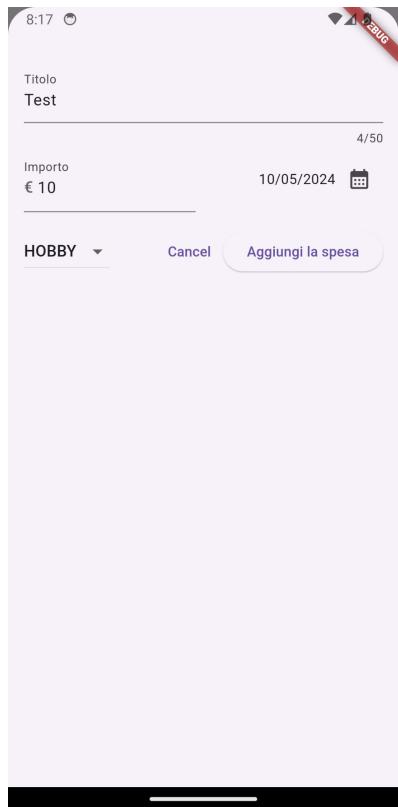
The chart

€0.00

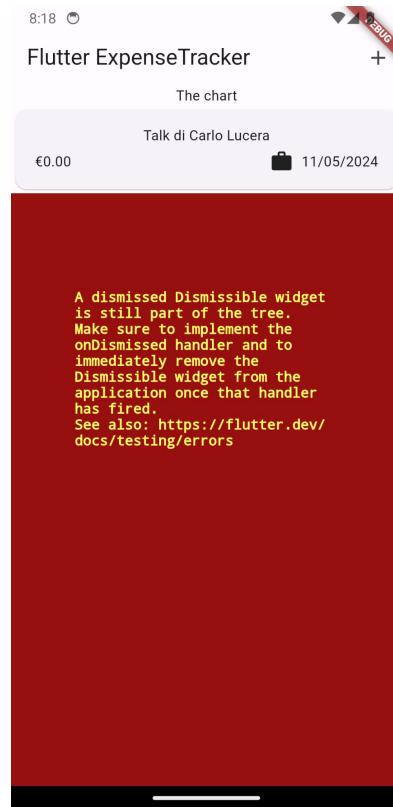
11/05/2024

Sembra tutto
funzionare
correttamente
ma...

Uso il widget Dismissible all'interno della ListView



Se provo ad
aggiungere un
nuovo
elemento...



...l'App va in crash!

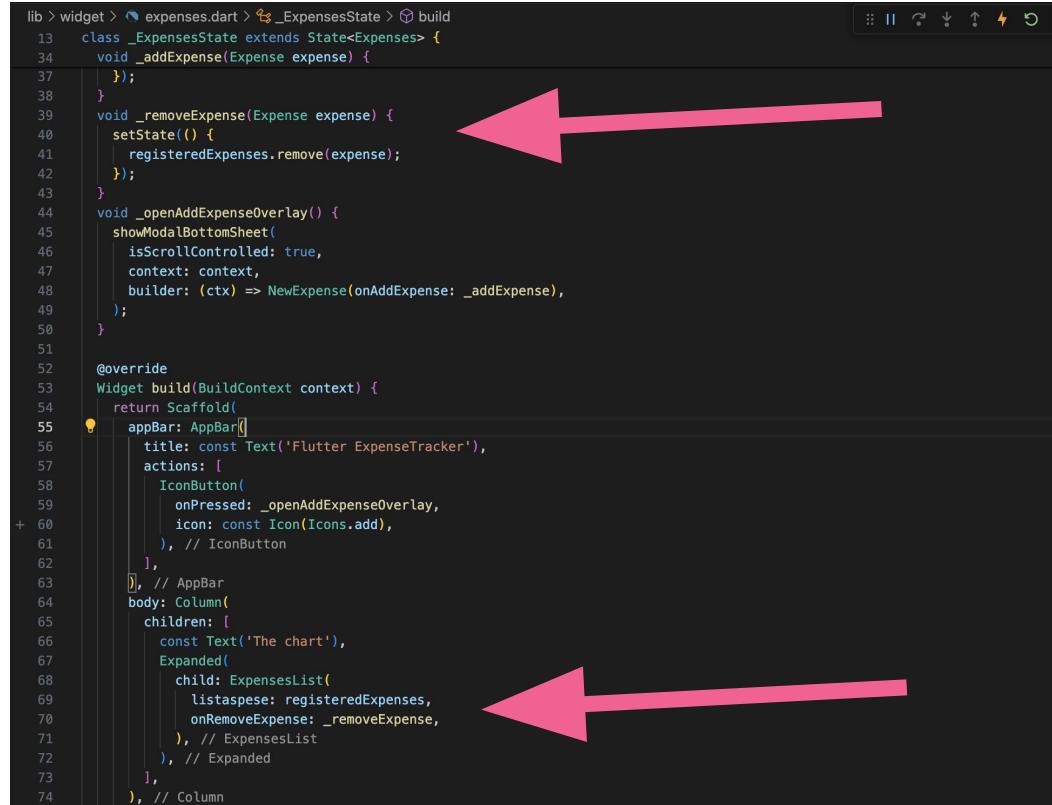
Il problema è che viene rimossa la lista solo in maniera grafica ma il dato all'interno del data model rimane presente quindi i dati diventano inconsistenti.
La soluzione? Aggiungere una funzionalità che consenta la rimozione anche dei dati!

Aggiungiamo il parametro onDismissed

```
expenses_list.dart ●  
lib > widget > expenses_list > expenses_list.dart > ...  
1 import 'package:flutter/material.dart';  
2 import 'package:lezione11/widget/expenses_list/expenses_item.dart';  
3 import 'package:lezione11/models/expense.dart';  
4  
5 class ExpensesList extends StatelessWidget {  
6   const ExpensesList({  
7     super.key,  
8     required this.listaspese,  
9     required this.onRemoveExpense,  
10   });  
11  
12   final List<Expense> listaspese;  
13   final void Function(Expense spesa) onRemoveExpense;  
14  
15   @override  
16   Widget build(BuildContext context) {  
17     return ListView.builder(  
18       itemCount: listaspese.length,  
19       itemBuilder: (ctx, index) => Dismissible(  
20         key: ValueKey(listaspese[index]),  
21         onDismissed: (direction) {  
22           onRemoveExpense(listaspese[index]);  
23         },  
24         child: ExpenseItem(  
25           listaspese[index],  
26         ), // ExpenseItem  
27         ), // Dismissible  
28       ); // ListView.builder  
29     }  
30   }  
31 }
```

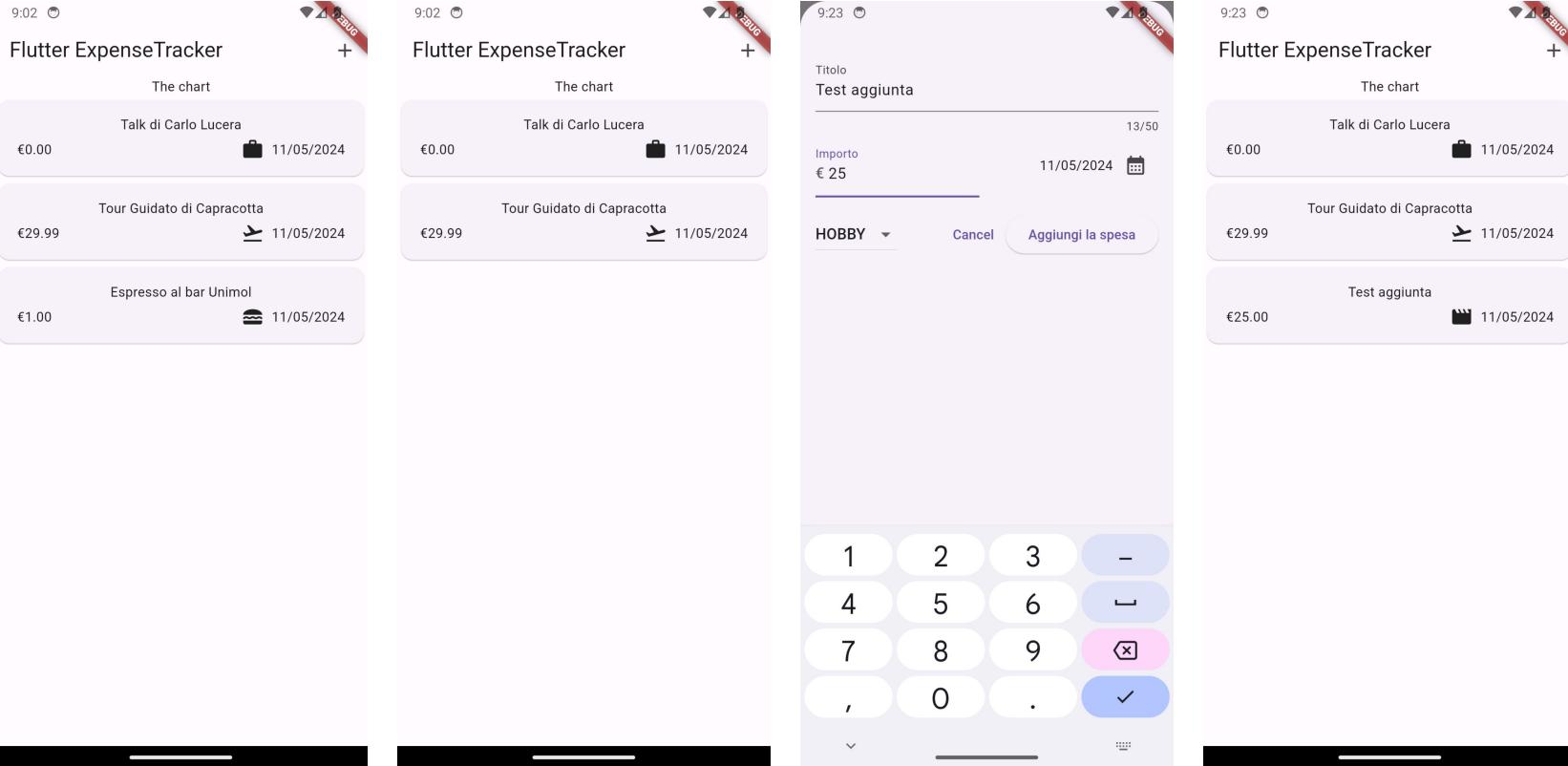


Adeguiamo anche il widget Expenses



```
lib > widget > expenses.dart > _ExpensesState > build
13   class _ExpensesState extends State<Expenses> {
14     void _addExpense(Expense expense) {
15       );
16     }
17     void _removeExpense(Expense expense) {
18       setState(() {
19         registeredExpenses.remove(expense);
20       });
21     }
22     void _openAddExpenseOverlay() {
23       showModalBottomSheet(
24         isScrollControlled: true,
25         context: context,
26         builder: (ctx) => NewExpense(onAddExpense: _addExpense),
27       );
28     }
29
30   @override
31   Widget build(BuildContext context) {
32     return Scaffold(
33       appBar: AppBar(
34         title: const Text('Flutter ExpenseTracker'),
35         actions: [
36           IconButton(
37             onPressed: _openAddExpenseOverlay,
38             icon: const Icon(Icons.add),
39           ), // IconButton
40           ],
41         ], // AppBar
42         body: Column(
43           children: [
44             const Text('The chart'),
45             Expanded(
46               child: ExpensesList(
47                 listaspe: registeredExpenses,
48                 onRemoveExpense: _removeExpense,
49               ), // ExpensesList
50             ), // Expanded
51           ],
52         ), // Column
53       ),
54     );
55   }
56 }
```

Risultato: Funziona!



Gestiamo una lista vuota e delle notifiche all'utente quando vengono fatte operazioni di cancellazione

Messaggio informativo se la lista è vuota (Widget Expenses)

```
  @override
  Widget build(BuildContext context) {
    Widget mainContent = const Center(
      child: Text('Non ci sono elementi nella tua lista della spesa!'),
    ); // Center
    if (registeredExpenses.isNotEmpty) {
      mainContent = ExpensesList(
        listaspese: registeredExpenses,
        onRemoveExpense: _removeExpense,
      );
    }
    return Scaffold(
      appBar: AppBar(
        title: const Text('Flutter ExpenseTracker'),
        actions: [
          IconButton(
            onPressed: _openAddExpenseOverlay,
            icon: const Icon(Icons.add),
          ), // IconButton
        ],
      ), // AppBar
      body: Column(
        children: [
          Expanded(
            child: mainContent,
          ), // Expanded
        ],
      ), // Column
    ); // Scaffold
```

SnackBar informativo con supporto per gli incerti

```
void _removeExpense(Expense expense) {
    final expenseIndex = registeredExpenses.indexOf(expense);

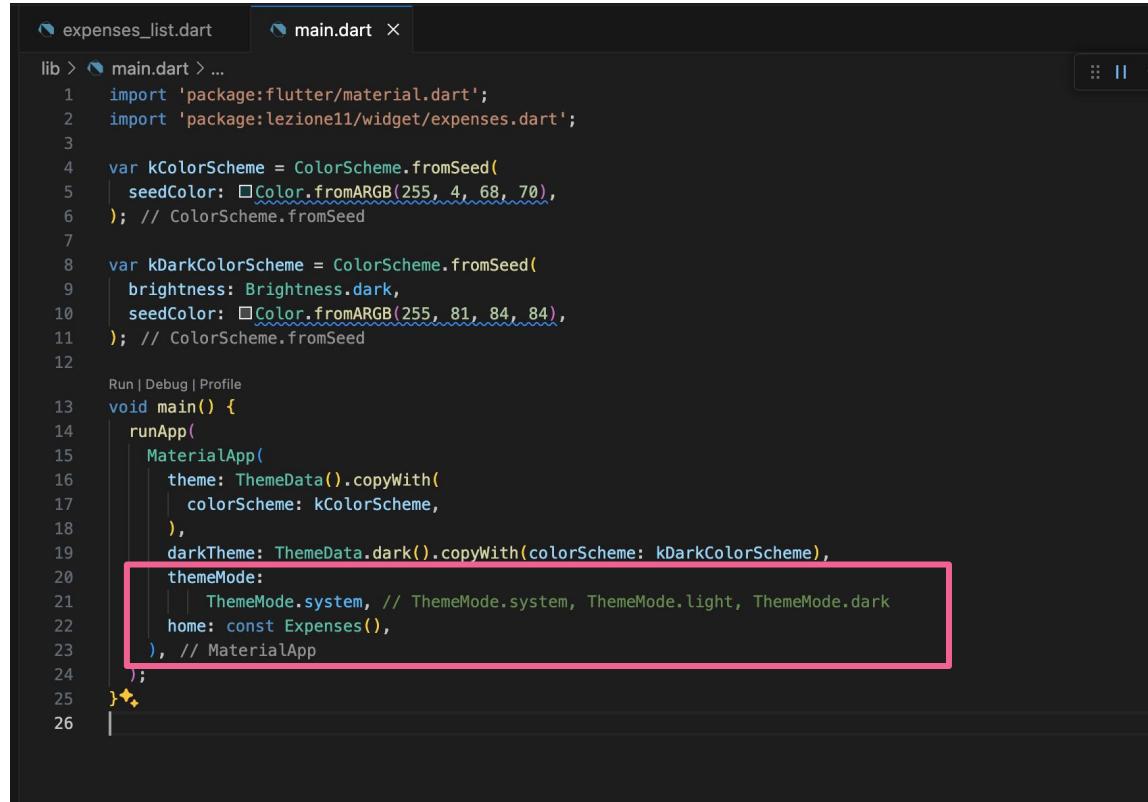
    setState(() {
        registeredExpenses.remove(expense);
    });

    ScaffoldMessenger.of(context)
        .clearSnackBars(); // rimuove snackbars precedenti
    ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            duration: const Duration(seconds: 3),
            content: const Text('Spesa Cancellata!'),
            action: SnackBarAction(
                label: 'Annulla ',
                onPressed: () {
                    setState(() {
                        registeredExpenses.insert(expenseIndex, expense);
                    });
                },
                ),
            ),
        );
}
}
```



Tema chiaro e dark,
gestiamoli entrambi

Temi Light e Dark



The screenshot shows a code editor with two tabs: 'expenses_list.dart' and 'main.dart'. The 'main.dart' tab is active, displaying the following Dart code:

```
lib > main.dart > ...
1 import 'package:flutter/material.dart';
2 import 'package:lezione11/widget/expenses.dart';
3
4 var kColorScheme = ColorScheme.fromSeed(
5   seedColor: Color.fromARGB(255, 4, 68, 70),
6 ); // ColorScheme.fromSeed
7
8 var kDarkColorScheme = ColorScheme.fromSeed(
9   brightness: Brightness.dark,
10  seedColor: Color.fromARGB(255, 81, 84, 84),
11 ); // ColorScheme.fromSeed
12
13 Run | Debug | Profile
14 void main() {
15   runApp(
16     MaterialApp(
17       theme: ThemeData().copyWith(
18         colorScheme: kColorScheme,
19       ),
20       darkTheme: ThemeData.dark().copyWith(colorScheme: kDarkColorScheme),
21       themeMode:
22         ThemeMode.system, // ThemeMode.system, ThemeMode.light, ThemeMode.dark
23         home: const Expenses(),
24     ), // MaterialApp
25   );
26 }
```

A red rectangular box highlights the 'themeMode' line of code, specifically the part where it is set to `ThemeMode.system`.



Quel fastidioso
ribbon di Debug...

Prima e dopo

11:25 ⚡ DEBUG
Flutter ExpenseTracker +

Talk di Carlo Lucera	€0.00	11/05/2024
Tour Guidato di Capracotta	€29.99	11/05/2024
Espresso al bar Unimol	€1.00	11/05/2024

```
void main() {
  runApp(
    MaterialApp(
      debugShowCheckedModeBanner: false,
      theme: ThemeData().copyWith(
        colorScheme: kColorScheme,
      ),
      darkTheme: ThemeData.dark().copyWith(colorScheme: kDarkColorScheme),
      themeMode:
        | | ThemeMode.system, // ThemeMode.system, ThemeMode.light, ThemeMode.dark
      home: const Expenses(),
    ),
  );
}
```

11:27 ⚡ DEBUG
Flutter ExpenseTracker +

Talk di Carlo Lucera	€0.00	11/05/2024
Tour Guidato di Capracotta	€29.99	11/05/2024
Espresso al bar Unimol	€1.00	11/05/2024

Esercizio

Nessun esercizio, semplicemente scaricate progetto.zip, riguardate il codice, provate a fare piccoli cambiamenti, acquisite dimestichezza con le nuove funzionalità.