Documentazione:

Main:

Importazioni:

dart

```
import 'package: shared_preferences/shared_preferences.dart';
import 'calendarpage. dart';
import 'package: flutter/material.dart';
import 'package: iconsax/iconsax.dart';
import 'dart: convert';
```

- `shared_preferences`: Utilizzato per salvare e recuperare i dati localmente.
- `calendarpage.dart`: Presumibilmente contiene la definizione di `MyCalendar`.
- `flutter/material.dart`: Framework principale per lo sviluppo di UI in Flutter.
- `iconsax`: Una libreria di icone.
- `dart:convert`: Utilizzato per la codifica e decodifica JSON.

Funzione Principale:

dart

```
void main() => runApp(const NavigationBarApp());
```

Avvia l'applicazione Flutter.

Classe `NavigationBarApp`

dart

```
class NavigationBarApp extends StatelessWidget {
  const NavigationBarApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(useMaterial3: true),
      home: const NavigationExample(),
    );
  }
```

}

Questa classe rappresenta l'applicazione principale e imposta il tema e la pagina iniziale dell'app.

Classe `NavigationExample`

dart

```
class NavigationExample extends StatefulWidget {
  const NavigationExample({super.key});
  @override
  State<NavigationExample> createState() => _NavigationExampleState();
}
```

Questa classe rappresenta il widget principale con stato che contiene la barra di navigazione.

Stato di `NavigationExample`

dart

```
class _NavigationExampleState extends State<NavigationExample> {
  int currentPageIndex = 0;
  List<String> dispositivi = [];
  List<Map<String, String>> personale = [];
  List<Map<String, dynamic>> eventi = [];
  @override
  void initState() {
    super.initState();
    _loadData();
  }
  Future<void> _loadData() async {
    final SharedPreferences prefs = await SharedPreferences.getInstance();
    setState(() {
      dispositivi = (prefs.getStringList('dispositivi') ?? []);
```

```
      personale = (jsonDecode(prefs.getString('personale') ?? '[]') as List).map((e) => Map<String, String>.from(e)).toList();
      eventi = (jsonDecode(prefs.getString('eventi') ?? '[]') as List).map((e) => Map<String, dynamic>.from(e)).toList();
    });
  }
  Future<void> _saveData() async {
    final SharedPreferences prefs = await SharedPreferences.getInstance();
    await prefs.setStringList('dispositivi', dispositivi);
    await prefs.setString('personale', jsonEncode(personale));
    await prefs.setString('eventi', jsonEncode(eventi));
  }
  void _addEvent(DateTime startDate, TimeOfDay startTime, DateTime endDate, TimeOfDay endTime, String title, List<String> selectedDispositivi, List<String> selectedPersonale) {
    setState(() {
      eventi.add({
        'startDate': startDate.toIso8601String(),
        'startTime': {'hour': startTime.hour, 'minute': startTime.minute},
        'endDate': endDate.toIso8601String(),
        'endTime': {'hour': endTime.hour, 'minute': endTime.minute},
        'title': title,
        'dispositivi': selectedDispositivi,
        'personale': selectedPersonale,
      });
      _saveData();
    });
  }
```

- `initState`: Inizializza lo stato caricando i dati salvati.
- `_loadData`: Carica i dati da `SharedPreferences`.
- `_saveData`: Salva i dati su `SharedPreferences`.
- `_addEvent`: Aggiunge un nuovo evento alla lista degli eventi e salva i dati.

Metodo `build`

```dart
 @override
 Widget build(BuildContext context) {
  final List<Widget> pages = <Widget>[
   // Home Page
   Scaffold(
    body: MyCalendar(
     eventi: eventi,
     dispositivi: dispositivi,
     personale: personale,
     onCreateEvent: _addEvent,
    ),
   ),

   **Devices Page**
   Scaffold(
    appBar: AppBar(
     backgroundColor: const Color.fromARGB(255, 132, 169, 140),
    ),
    body: dispositivi.isEmpty
       ? Center(
          child: Text(
           'Nessun dispositivo aggiunto',
           style: TextStyle(color: const Color.fromARGB(255, 0, 0, 0)),
          ),
         )
       : ListView.builder(
          itemCount: dispositivi.length,
          itemBuilder: (context, index) {
```

```
return Container(
  margin: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 16.0),
  padding: const EdgeInsets.all(16.0),
  decoration: BoxDecoration(
    color: Colors.white,
    borderRadius: BorderRadius.circular(8.0),
    boxShadow: [
      BoxShadow(
        color: Colors.black12,
        blurRadius: 8.0,
        spreadRadius: 1.0,
      ),
    ],
  ),
  child: Row(
    children: [
      Flexible(
        child: Text(
          dispositivi[index],
          style: TextStyle(
            color: Colors.black,
            fontSize: 18.0,
          ),
        ),
      ),
      IconButton(
        icon: Icon(Iconsax.box_remove, color: Colors.red),
        onPressed: () {
          setState(() {
            dispositivi.removeAt(index);
            _saveData();
          });
```

```
                },
              ),
            ],
          ),
        );
      },
    ),
),
```

Personnel Page

```
Scaffold(
  appBar: AppBar(
    title: Text('Personale'),
    backgroundColor: const Color.fromARGB(255, 132, 169, 140),
  ),
  body: personale.isEmpty
      ? Center(
          child: Text(
            'Nessun personale aggiunto',
            style: TextStyle(color: const Color.fromARGB(255, 0, 0, 0)),
          ),
        )
      : ListView.builder(
          itemCount: personale.length,
          itemBuilder: (context, index) {
            final person = personale[index];
            return Container(
              margin: const EdgeInsets.symmetric(vertical: 8.0, horizontal: 16.0),
              padding: const EdgeInsets.all(16.0),
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(8.0),
```

```
  boxShadow: [
    BoxShadow(
      color: Colors.black12,
      blurRadius: 8.0,
      spreadRadius: 1.0,
    ),
  ],
),
child: Row(
  mainAxisAlignment: MainAxisAlignment.spaceBetween,
  children: [
    Flexible(
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            'Nome: ${person['nome']!}',
            style: TextStyle(
              color: Colors.blue,
              fontSize: 18.0,
            ),
          ),
          Text(
            'Professione: ${person['professione']!}',
            style: TextStyle(
              color: Colors.green,
              fontSize: 16.0,
            ),
          ),
          Text(
            'Età: ${person['eta']!}',
            style: TextStyle(
```

```
                    color: Colors.red,
                    fontSize: 14.0,
                  ),
                ),
              ],
            ),
          ),
          IconButton(
            splashColor: Color.fromARGB(255, 132, 169, 140),
            icon: Icon(Iconsax.profile_delete,
              color: Colors.red,
              size: 30,
            ),
            onPressed: () {
              setState(() {
                personale.removeAt(index);
                _saveData();
              });
            },
          ),
        ],
      ),
    );
  },
 ),
```

Settings Page

```
];

return Scaffold(
```

```
bottomNavigationBar: NavigationBar(
  height: 90,
  onDestinationSelected: (int index) {
    setState(() {
      currentPageIndex = index;
    });
  },
  backgroundColor: const Color.fromARGB(255, 132, 169, 140),
  indicatorColor: const Color.fromARGB(255, 82, 121, 111),
  selectedIndex: currentPageIndex,
  destinations: const <Widget>[
    NavigationDestination(
      icon: Icon(Iconsax.home_2),
      label: 'Home',
    ),
    NavigationDestination(
      icon: Icon(Iconsax.box),
      label: 'Dispositivi',
    ),
    NavigationDestination(
      icon: Icon(Iconsax.personalcard),
      label: 'Personale',
    ),

  ],
),
body: pages[currentPageIndex],
floatingActionButton: currentPageIndex == 1 || currentPageIndex == 2
    ? FloatingActionButton(
        backgroundColor: Color.fromARGB(255, 132, 169, 140),
        splashColor: Color.fromARGB(255, 132, 169, 140),
        onPressed: () {
```

```
showDialog(
  context: context,
  builder: (BuildContext context) {
    if (currentPageIndex == 1) {

      Dialog for adding a device
      String newDevice

= '';

      return AlertDialog(
        title: Text('Aggiungi Dispositivo'),
        content: TextField(
          onChanged: (valore) {
            newDevice = valore;
          },
          decoration: InputDecoration(
            hintText: 'Nome del dispositivo',
          ),
        ),
        actions: <Widget>[
          TextButton(
            child: Text('Annulla'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          TextButton(
            child: Text('Aggiungi'),
            onPressed: () async {
              setState(() {
                dispositivi.add(newDevice);
                _saveData();
```

```
          });
          Navigator.of(context).pop();
        },
      ),
    ],
  );
} else if (currentPageIndex == 2) {
```

Dialog for adding personnel

```
  String nome = '';
  String professione = '';
  String eta = '';
  return AlertDialog(
    title: Text('Aggiungi Personale'),
    content: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        TextField(
          onChanged: (valore) {
            nome = valore;
          },
          decoration: InputDecoration(
            hintText: 'Nome',
          ),
        ),
        TextField(
          onChanged: (valore) {
            professione = valore;
          },
          decoration: InputDecoration(
            hintText: 'Professione',
          ),
```

```
    ),
    TextField(
      onChanged: (valore) {
        eta = valore;
      },
      decoration: InputDecoration(
        hintText: 'Età',
      ),
    ),
  ],
),
actions: <Widget>[
  TextButton(
    child: Text('Annulla'),
    onPressed: () {
      Navigator.of(context).pop();
    },
  ),
  TextButton(
    child: Text('Aggiungi'),
    onPressed: () {
      setState(() {
        personale.add({
          'nome': nome,
          'professione': professione,
          'eta': eta,
        });
        _saveData();
      });
      Navigator.of(context).pop();
    },
  ),
```

```
                    ],
                  );
                }
                return Container();
              },
            );
          },
          child: Icon(Icons.add),
        )
      : null,
  );
 }
}
```

- `pages`: Lista delle pagine visualizzate in base alla selezione della barra di navigazione.
- `NavigationBar`: Barra di navigazione nella parte inferiore dello schermo con destinazioni predefinite.
- `FloatingActionButton`: Bottone per aggiungere dispositivi o personale, a seconda della pagina corrente.

Descrizione delle Pagine

1. **Home Page**:
   - Mostra un calendario con gli eventi.

2. **Devices Page**:
   - Mostra una lista di dispositivi con la possibilità di aggiungere o rimuovere dispositivi.

3. **Personnel Page**:
   - Mostra una lista di personale con la possibilità di aggiungere o rimuovere personale.

Dialogs

- **Aggiungi Dispositivo**:

  - Permette di aggiungere un nuovo dispositivo.

- **Aggiungi Personale**:

  - Permette di aggiungere un nuovo membro del personale con nome, professione ed età.

Conclusione

Il codice implementa un'applicazione Flutter con navigazione tra diverse pagine utilizzando una barra di navigazione. Consente di gestire eventi, dispositivi e personale, con persistenza dei dati utilizzando `SharedPreferences`.

Calendar Page:

Ecco la documentazione del codice Dart/Flutter che hai fornito. Questo codice definisce un widget `MyCalendar` che visualizza un calendario con eventi, consentendo all'utente di aggiungere, visualizzare e eliminare eventi.

Importazioni

dart

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

- `flutter/material.dart`: Framework principale per lo sviluppo di UI in Flutter.
- `table_calendar/table_calendar.dart`: Pacchetto per la visualizzazione di calendari.

Classe `MyCalendar`

Questa classe è un widget di stato che rappresenta un calendario con eventi.

Costruttore

```dart
MyCalendar({
  required this.eventi,
  required this.dispositivi,
  required this.personale,
  required this.onCreateEvent,
});
```

- `eventi`: Lista di eventi.
- `dispositivi`: Lista di dispositivi.
- `personale`: Lista di personale.
- `onCreateEvent`: Funzione callback per la creazione di un evento.

Stato di `MyCalendar`

La classe `_MyCalendarState` gestisce lo stato del widget `MyCalendar`.

Variabili di Stato

```dart
DateTime selectedDay = DateTime.now();
CalendarFormat _calendarFormat = CalendarFormat.month;
```

- `selectedDay`: Giorno selezionato nel calendario.
- `_calendarFormat`: Formato corrente del calendario (mese o settimana).

Metodi

- `_onDaySelected`: Metodo chiamato quando un giorno viene selezionato nel calendario.

  ```dart
  ```

```
void _onDaySelected(DateTime selectedDay, DateTime focusedDay) {
  setState(() {
    this.selectedDay = selectedDay;
  });
  _showEventsDialog(selectedDay);
}
```

-- `_showEventsDialog`: Mostra un dialogo con gli eventi del giorno selezionato.

```dart
void _showEventsDialog(DateTime day) {
  showDialog(
    context: context,
    builder: (context) {
      final events = widget.eventi.where((event) {
        final startDate = DateTime.parse(event['startDate']);
        final endDate = DateTime.parse(event['endDate']);
        return isSameDay(startDate, day) ||
            isSameDay(endDate, day) ||
            (startDate.isBefore(day) && endDate.isAfter(day));
      }).toList();
      return AlertDialog(
        title: Text("Eventi del giorno"),
        content: SingleChildScrollView(
          child: events.isEmpty
              ? Text("Nessun evento")
              : Column(
                  mainAxisSize: MainAxisSize.min,
                  children: events.map((event) {
                    return ListTile(
                      title: Text(event['title']),
                      subtitle: Column(
                        crossAxisAlignment: CrossAxisAlignment.start,
```

```
                    children: [
                      Text("Inizio: ${event['startTime']}"),
                      Text("Fine: ${event['endTime']}"),
                      Text("Dispositivi usati: ${event['dispositivi'].join(', ')}"),
                      Text("Personale coinvolto: ${event['personale'].join(', ')}"),
                    ],
                  ),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () {
                      _deleteEvent(event);
                      Navigator.of(context).pop();
                    },
                  ),
                );
              }).toList(),
            ),
          ),
          actions: [
            TextButton(
              child: Text("Chiudi"),
              onPressed: () {
                Navigator.of(context).pop();
              },
            ),
          ],
        );
      },
    );
  }
```

-- `_deleteEvent`: Elimina un evento dalla lista degli eventi.

```dart
void _deleteEvent(Map<String, dynamic> event) {
  setState(() {
    widget.eventi.remove(event);
  });
}
```

-- `_showAddEventDialog`: Mostra un dialogo per aggiungere un nuovo evento.

```dart
void _showAddEventDialog() {
  DateTime startDate = selectedDay;
  DateTime endDate = selectedDay;
  TimeOfDay startTime = TimeOfDay.now();
  TimeOfDay endTime = TimeOfDay.now();
  String title = '';
  List<String> selectedDispositivi = [];
  List<String> selectedPersonale = [];

  showDialog(
    context: context,
    builder: (context) {
      return StatefulBuilder(
        builder: (context, setState) {
          return AlertDialog(
            title: Text("Crea Evento"),
            content: Container(
              width: double.maxFinite,
              child: ListView(
                shrinkWrap: true,
                children: [
                  TextField(
                    onChanged: (valore) {
```

```
    title = valore;
  },
  decoration: InputDecoration(hintText: 'Titolo'),
),
SizedBox(height: 16),
Text("Data di Inizio:"),
SizedBox(height: 8),
ElevatedButton(
  onPressed: () async {
    final DateTime? picked = await showDatePicker(
      context: context,
      initialDate: startDate,
      firstDate: DateTime(2010),
      lastDate: DateTime(2030),
    );
    if (picked != null && picked != startDate) {
      setState(() {
        startDate = picked;
      });
    }
  },
  child: Text("${startDate.toLocal()}".split(' ')[0]),
),
SizedBox(height: 16),
Text("Ora di Inizio:"),
SizedBox(height: 8),
ElevatedButton(
  onPressed: () async {
    final TimeOfDay? picked = await showTimePicker(
      context: context,
      initialTime: startTime,
    );
    if (picked != null && picked != startTime) {
      setState(() {
```

```
        startTime = picked;
      });
    }
  },
  child: Text("${startTime.format(context)}"),
),
SizedBox(height: 16),
Text("Data di Fine:"),
SizedBox(height: 8),
ElevatedButton(
  onPressed: () async {
    final DateTime? picked = await showDatePicker(
      context: context,
      initialDate: endDate,
      firstDate: DateTime(2010),
      lastDate: DateTime(2030),
    );
    if (picked != null && picked != endDate) {
      setState(() {
        endDate = picked;
      });
    }
  },
  child: Text("${endDate.toLocal()}".split(' ')[0]),
),
SizedBox(height: 16),
Text("Ora di Fine:"),
SizedBox(height: 8),
ElevatedButton(
  onPressed: () async {
    final TimeOfDay? picked = await showTimePicker(
      context: context,
      initialTime: endTime,
    );
```

```
    if (picked != null && picked != endTime) {
      setState(() {
        endTime = picked;
      });
    }
  },
  child: Text("${endTime.format(context)}"),
),
SizedBox(height: 16),
Text("Dispositivi:"),
Container(
  height: 100,
  child: ListView(
    children: widget.dispositivi.map((dispositivo) {
      return CheckboxListTile(
        title: Text(dispositivo),
        value: selectedDispositivi.contains(dispositivo),
        onChanged: (bool? value) {
          setState(() {
            if (value == true) {
              selectedDispositivi.add(dispositivo);
            } else {
              selectedDispositivi.remove(dispositivo);
            }
          });
        },
      );
    }).toList(),
  ),
),
SizedBox(height: 16),
Text("Personale:"),
Container(
  height: 100,
```

```
                child: ListView(
                  physics: const AlwaysScrollableScrollPhysics(),
                  children: widget.personale.map((person) {
                    return CheckboxListTile(
                      title: Text(person['nome']!),
                      value: selectedPersonale.contains(person['nome']!),
                      onChanged: (bool? value) {
                        setState(() {
                          if (value == true) {
                            selectedPersonale.add(person['nome']!);
                          } else {
                            selectedPersonale.remove(person['nome']!);
                          }
                        });
                      },
                    );
                  }).toList(),
                ),
              ),
            ],
          ),
        ),
        actions: [
          TextButton(
            child: Text("Annulla"),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ),
          TextButton(
            child: Text("Crea"),
            onPressed: () {
              widget.onCreateEvent(startDate, startTime, endDate, endTime, title, selectedDispositivi, selectedPersonale);
```

```
                Navigator.of(context).pop();
              },
            ),
          ],
        );
      },
    );
  },
 );
}
```

Metodo `build`

Il metodo `build` costruisce l'interfaccia utente del widget.

```dart
@override
Widget build(BuildContext context) {
 return Scaffold(
  appBar: AppBar(
   backgroundColor: Color.fromARGB(255, 132, 169, 140),
   title: const Text(
    'Schedle',
    style: TextStyle(
     fontSize: 30.0,
     fontWeight: FontWeight.bold,
    ),
   ),
  ),
  body: ClipRRect(
   child: calendario(),
  ),
  floatingActionButton: FloatingActionButton(
   backgroundColor: Color.fromARGB(255, 82, 121, 111),
```

```
      child: Icon(Icons.add),
      onPressed: () {
        _showAddEventDialog();
      },



 ),
  );
}
```

Metodo `calendario`

Costruisce il widget del calendario.

```dart
Widget calendario() {
  final DateTime today = DateTime.now();
  final eventiGiornalieri = widget.eventi.where((event) {
    final startDate = DateTime.parse(event['startDate']);
    final endDate = DateTime.parse(event['endDate']);
    return isSameDay(startDate, today) ||
        isSameDay(endDate, today) ||
        (startDate.isBefore(today) && endDate.isAfter(today));
  }).toList();

  return Column(
    children: [
      Padding(padding: EdgeInsets.all(0)),
      Container(
        child: TableCalendar(
          availableCalendarFormats: const {
            CalendarFormat.month: 'Settimana',
            CalendarFormat.week: 'Mese',
          },
```

```
    calendarStyle: CalendarStyle(
      todayDecoration: BoxDecoration(
        shape: BoxShape.circle,
        color: Color.fromARGB(255, 82, 121, 111),
      ),
      selectedDecoration: BoxDecoration(
        shape: BoxShape.circle,
        color: Color.fromARGB(255, 132, 169, 140),
      ),
    ),
    focusedDay: selectedDay,
    firstDay: DateTime.utc(2010, 1, 1),
    lastDay: DateTime.utc(2030, 1, 1),
    selectedDayPredicate: (day) => isSameDay(day, selectedDay),
    onDaySelected: _onDaySelected,
    calendarFormat: _calendarFormat,
    onFormatChanged: (format) {
      setState(() {
        _calendarFormat = format;
      });
    },
    eventLoader: (day) {
      return widget.eventi.where((event) {
        final startDate = DateTime.parse(event['startDate']);
        final endDate = DateTime.parse(event['endDate']);
        return isSameDay(startDate, day) ||
            isSameDay(endDate, day) ||
            (startDate.isBefore(day) && endDate.isAfter(day));
      }).toList();
    },
  ),
),
Expanded(
  child: Padding(
```

```
padding: const EdgeInsets.symmetric(vertical: 8.0),
child: Container(
  decoration: BoxDecoration(
    borderRadius: BorderRadius.circular(20),
    color: Color.fromARGB(255, 132, 169, 140),
  ),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Center(
        child: Padding(
          padding: const EdgeInsets.all(8),
          child: Text(
            "Eventi del giorno",
            style: TextStyle(
              fontSize: 18,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
        ),
      ),
    ),
    if (eventiGiornalieri.isNotEmpty)
      Expanded(
        child: ListView.builder(
          padding: EdgeInsets.all(5.0),
          itemCount: eventiGiornalieri.length,
          itemBuilder: (context, index) {
            final event = eventiGiornalieri[index];
            return Padding(
              padding: const EdgeInsets.only(bottom: 8.0),
              child: Container(
                decoration: BoxDecoration(
                  color: Color.fromARGB(255, 82, 121, 111),
```

```
                    borderRadius: BorderRadius.circular(20),
                  ),
                  padding: EdgeInsets.all(20.0),
                  child: Text(
                    "${event['title']}: dalle ore ${event['startTime']} alle ore ${event['endTime']}",
                    style: TextStyle(fontSize: 16, color: Colors.white),
                  ),
                ),
              );
            },
          ),
        )
      else
        Expanded(
          child: Center(
            child: Text(
              "Nessun evento oggi",
              style: TextStyle(
                fontSize: 16,
                color: Colors.white,
              ),
            ),
          ),
        ),
    ],
  ),
),
),
),
),
Padding(padding: EdgeInsets.all(35.0)),
],
);
}
```

Conclusione

Il widget `MyCalendar` fornisce una visualizzazione del calendario che consente agli utenti di selezionare i giorni, visualizzare eventi giornalieri e aggiungere nuovi eventi. Utilizza `TableCalendar` per la visualizzazione del calendario e `AlertDialog` per la gestione dell'interazione con l'utente. La funzione `onCreateEvent` viene utilizzata per gestire la creazione di nuovi eventi al di fuori di questo widget.