

Build a Flutter Form with Custom Input Types

 blog.bam.tech/developer-news/build-a-flutter-form-with-custom-input-types

Dealing with forms is a recurring topic in web and mobile application development. Nevertheless they can sometimes be tedious to manage. Especially when it comes to including several types of user input. Google's Flutter framework provides an elegant way to deal with forms. Let's see together how to use it on purpose.

What you will learn

You will learn one way to implement your own FormField library in Flutter, including multiple user input types such as Switch, ToggleButtons, Multiselection and Date.

Building Form In Flutter

In a recent Flutter project I had to implement some complex forms with custom input fields such as toggles or dates. The Flutter framework provides a pretty good template to manage form in your project. This includes validation and submission both at form and field level or decoupling style with an InputDecorator. But if the doc is quite clear concerning the implementation of basic forms with TextFormField (check [the official documentation](#) or [this good article from Coding With Joe](#)), there isn't a lot of information about custom FormFields.

I will now share with you how my team implemented its own FormField library.

Learning by example

Together we will build a sign up form for a fake dating app. Here is what the final result looks like:

Let's start with an initial implementation of this form composed only of a TextFormField to retrieve the user's name.

```
// main.dart

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Dating App - Signup Form',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Signup Form'),
    );
  }
}
```

```
}
```

```
class SignupUser {  
  String name;  
  SignupUser({  
    this.name,  
  });  
  Map<String, dynamic> toJson() => {  
    'name': name,  
  };  
}
```

```
class MyHomePage extends StatefulWidget {  
  MyHomePage({Key key, this.title}) : super(key: key);  
  final String title;  
  @override  
  _MyHomePageState createState() => _MyHomePageState();  
}
```

```
class _MyHomePageState extends State {  
  final GlobalKey _formKey = GlobalKey();  
  final _formResult = SignupUser();  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(widget.title),  
      ),  
      body: SafeArea(  
        top: false,  
        bottom: false,  
        child: Form(  
          key: _formKey,  
          autovalidate: true,  
          child: ListView(  
            padding: const EdgeInsets.symmetric(horizontal: 16.0),  
            children: [  
              TextFormField(  
                decoration: const InputDecoration(  
                  hintText: 'Enter your name',  
                  labelText: 'Name',  
                ),  
                inputFormatters: [LengthLimitingTextInputFormatter(30)],  
                initialValue: _formResult.name,  
                validator: (userName) {  
                  if (userName.isEmpty) {  
                    return 'Name is required';  
                  }  
                  if (userName.length < 3) {  
                    return 'Name is too short';  
                  }  
                  return null;  
                },  
                textInputAction: TextInputAction.next,  
              ),  
            ],  
          ),  
        ),  
      ),  
    );  
  }  
}
```

```

        autofocus: true,
        onSave: (userName) {
          _formResult.name = userName;
        },
      ),
    ],
  ),
),
),
floatingActionButton: FloatingActionButton(
  onPressed: _submitForm,
  tooltip: 'Save',
  child: Icon(
    Icons.check,
    size: 36.0,
  ),
),
);
}

void _submitForm() {
  final FormState form = _formKey.currentState;
  if (form.validate()) {
    form.save();
    print('New user saved with signup data:\n');
    print(_formResult.toJson());
  }
}
}

```

TextFormField is a common Flutter widget. Taking example on its implementation I will now add new types of form fields.

SwitchFormField

Let's start with a simple one, a SwitchFormField based on Flutter Switch Widget. This field will come at the end of the form to acknowledge that the user agree with the App's ethics rules.

// mySwitchFormField.dart

```

class MySwitchFormField extends FormField<bool> {
  MySwitchFormField({
    Key key,
    bool initialValue, // Initial field value
    this.decoration =
      const InputDecoration(), // A BoxDecoration to style the field FormFieldSetter

    onSave, // Method called when when the form is saved FormFieldValidator

    validator, // Method called for validation

    this.onChangeed, // Method called whenever the value changes
  }) : super(
    key: key,
    initialValue: initialValue,
    decoration: decoration,
    validator: validator,
    onSave: onSave,
    onChangeed: onChangeed,
  );
}

```

```

this.constraints =
    const BoxConstraints(), // A BoxConstraints to set the switch size
}) : assert(decoration != null),
    assert(initialValue != null),
    assert(constraints != null),
    super(
        key: key,
        onSaved: onSaved,
        initialValue: initialValue,
        validator: validator,
        builder: (FormFieldState field) {
            final InputDecoration effectiveDecoration =
                decoration.applyDefaults(
                    Theme.of(field.context).inputDecorationTheme,
                );
            return InputDecorator(
                decoration:
                    effectiveDecoration.copyWith(errorText: field.errorText),
                isEmpty: field.value == null,
                child: Row(
                    children: [
                        ConstrainedBox(
                            constraints: constraints,
                            child: Switch(
                                value: field.value,
                                onChanged: field.didChange,
                            ),
                        ),
                    ],
                ),
            );
        },
    );
final ValueChanged onChanged;
final InputDecoration decoration;
final BoxConstraints constraints;
@override
FormFieldState<bool> createState() => _MySwitchFormFieldState();
}

```

```

class _MySwitchFormFieldState extends FormFieldState<bool> {
    @override
    MySwitchFormField get widget => super.widget;
    @override
    void didChange(bool value) {
        super.didChange(value);
        if (widget.onChanged != null) {
            widget.onChanged(value);
        }
    }
}

```

Then use it in the main form ListView like this:

```

// main.dart

// ...
class SignupUser {

  // ...
  bool ethicsAgreement;
  SignupUser({

    // ...
    this.ethicsAgreement = false,
  });
  Map<String, dynamic> toJson() => {

    // ...
    'ethicsAgreement': ethicsAgreement,
  };
}

// ...

class _MyHomePageState extends State {

  // ...
  @override
  Widget build(BuildContext context) {
    return Scaffold(

      // ...
      body: SafeArea(

        // ...
        child: Form(
          key: _formKey,
          autovalidate: true,
          child: ListView(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            children: [
              // ...
              SizedBox(height: 8.0),
              MySwitchFormField(
                decoration: InputDecoration(
                  labelText: 'Ethics agreement',
                  hintText: null,
                ),
                initialValue: _formResult.ethicsAgreement,
                validator: (userHasAgreedWithEthics) =>
                  userHasAgreedWithEthics == false
                    ? 'Please agree with ethics'
                    : null,
                onSave: (userHasAgreedWithEthics) {
                  _formResult.ethicsAgreement = userHasAgreedWithEthics;
                },
              ),
            ],
          ),
        ),
      ),
    );
  }
}

```

```

    ],
  ),
),
),
// ...
);
}

// ...
}

```

We can see that building a custom `FormField` consists in extending the Flutter basic `FormField` widget and then passing your specific user input `Widget` as a child, wrapped by an `InputDecorator` `Widget`. If you wonder why extending a `FormField` instead of just use the `FormField` as a wrapper as it is usually the case with Flutter composition pattern, I've just aligned myself with what Flutter offers already with `TextFormField` and `DropDownFormField` widgets.

Anyway, leaning on a `FormField` and an `InputDecorator` provide us way to:

- Set the field style (including error, focus, hint etc.) with an `InputDecoration`
- Handle validation at the field level with the `validator` method
- Handle input value submission with the `onSaved` method
- Add a custom `onChanged` behavior on user inputs with the `onChanged` method (in addition to the internal `onChanged` behavior of the stateful input widget)
- Handle focus with the `focusNode` attribute (this part is a bit more tricky and will be explained at the end)

On the other hand you keep the hand on what your user input looks like and how the user interact with it.

ToggleButtonsFormField

Let's add a gender field following the same pattern:

```
// myToggleButtonsFormField.dart
```

```

class MyToggleButtonsFormField<T> extends FormField<T> {
  MyToggleButtonsFormField({
    Key key,
    this.initialValue, // Initial selected option

    @required this.items, // Available options

    @required this.itemBuilder, // Widget builder for an option

    @required this.selectedItemBuilder, // Widget builder for the selected option
  }) : super(key: key, initialValue: this.initialValue);
}

```

```

    this.decoration = const InputDecoration(),
    this.onChanged,
    FormFieldSetter onSave,
    FormFieldValidator validator,
  }) : assert(decoration != null),
      assert(items != null),
      assert(itemBuilder != null),
      assert(selectedItemBuilder != null),
      assert(initialValue == null || items.contains(initialValue)),
      super(
        key: key,
        onSave: onSave,
        initialValue: initialValue,
        validator: validator,
        builder: (FormFieldState field) {
          final InputDecoration effectiveDecoration =
            decoration.applyDefaults(
              Theme.of(field.context).inputDecorationTheme,
            );
          return InputDecorator(
            decoration:
              effectiveDecoration.copyWith(errorText: field.errorText),
            child: MyToggleButtons(
              items: items,
              value: field.value,
              itemBuilder: itemBuilder,
              selectedItemBuilder: selectedItemBuilder,
              onPressed: field.didChange,
            ),
          );
        },
      );
  final List<T> items;
  final ValueChanged<T> onChanged;
  final T initialValue;
  final Widget Function(BuildContext, T) itemBuilder;
  final Widget Function(BuildContext, T) selectedItemBuilder;
  final InputDecoration decoration;
  @override
  _MyToggleButtonsFormFieldState<T> createState() =>
    _MyToggleButtonsFormFieldState<T>();
}

class _MyToggleButtonsFormFieldState<T> extends FormFieldState<T> {
  @override
  MyToggleButtonsFormField<T> get widget => super.widget;
  @override
  void didChange(T value) {
    super.didChange(value);
    if (widget.onChanged != null) {
      widget.onChanged(value);
    }
  }
}

```

```

// main.dart

// ...

enum Gender {
  Male,
  Female,
  Other,
}

class SignupUser {

  // ...
  Gender gender;
  SignupUser({

    // ...
    this.gender,
  });
  Map<String, dynamic> toJson() => {

    // ...
    'gender': gender.toString(),
  };
}

// ...
class _MyHomePageState extends State {

  // ...
  @override
  Widget build(BuildContext context) {
    return Scaffold(

      // ...
      body: SafeArea(

        // ...
        child: Form(
          key: _formKey,
          autovalidate: true,
          child: ListView(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            children: [

              // ...
              SizedBox(height: 8.0),
              MyToggleButtonsFormField<Gender>(
                decoration: InputDecoration(
                  labelText: 'Gender',
                ),
                initialValue: _formResult.gender,
                items: Gender.values,
                itemBuilder: (BuildContext context, Gender genderItem) =>

```



```

        Text(describeEnum(genderItem)),
        selectedItemBuilder: (BuildContext context, Gender genderItem) =>
        Text(describeEnum(genderItem)),
        validator: (gender) =>
        gender == null ? 'Gender is required' : null,
        onSave: (gender) {
        _formResult.gender = gender;
        },
        borderRadius: BorderRadius.all(Radius.circular(5.0)),
    ),
  ],
),
),
),
// ...
);
}

// ...
}

```

The input Widget is a custom implementation of the [ToggleButtons](#). I build my own widget as the Flutter ToggleButtons widget takes as input a list of boolean while mines takes only one typed option at time, and expands the buttons. You can find the complete implementation in the DartPad snippet above. What is important to notice is that we have just build a new FormField type based on a custom Widget. In fact by extending the formField widget (or use it as a wrapper) you are now able to build any form field you want. Following are some other examples.

MultiselectionFormField

This widget, more complex, combines a [DropDownButton](#), a list of stateful [CheckBox](#) and a list of [Chip](#) to retrieve the user's main interests.

// myMultiSelectionFormField.dart

```

class MyMultiSelectionFormField<T> extends FormField<List<T>> {
  MyMultiSelectionFormField({
    Key key,
    @required List<T> initialValues, // List of initial selected options

    @required List<T> options, // List of available options

    @required Widget Function(T) titleBuilder, // Widget builder for an option in the dropdown
    menu

    @required Widget Function(T) chipLabelBuilder, // Widget builder for the selected options
    in the chipList Widget

    hint, // A placeholder widget that is displayed by the dropdown button
    this.decoration = const InputDecoration(),
    this.onChanged,

```

```

    FormFieldSetter<List> onSaved,
    FormFieldValidator<List> validator,
  }) : assert(options == null ||
    options.isEmpty ||
    initialValues == null ||
    initialValues.every((value) =>
      options.where((T option) {
        return option == value;
      }).length ==
      1)),
    assert(decoration != null),
    super(
      key: key,
      onSaved: onSaved,
      initialValue: initialValues,
      validator: validator,
      builder: (FormFieldState<List> field) {
        final InputDecoration effectiveDecoration =
          decoration.applyDefaults(
            Theme.of(field.context).inputDecorationTheme,
          );
        return InputDecorator(
          decoration:
            effectiveDecoration.copyWith(errorText: field.errorText),
          isEmpty: field.value.isEmpty,
          child: MyMultiSelectionField(
            values: field.value,
            options: options,
            titleBuilder: titleBuilder,
            chipLabelBuilder: chipLabelBuilder,
            hint: field.value.isNotEmpty ? hint : null,
            onChanged: field.didChange,
          ),
        );
      },
    );
  final ValueChanged<List> onChanged;
  final InputDecoration decoration;
  @override
  _MyMultiSelectionFormFieldState<T> createState() =>
    _MyMultiSelectionFormFieldState();
}

class _MyMultiSelectionFormFieldState<T> extends FormFieldState<List<T>> {
  @override
  MyMultiSelectionFormField<T> get widget => super.widget;
  @override
  void didChange(List values) {
    super.didChange(values);
    if (widget.onChanged != null) {
      widget.onChanged(values);
    }
  }
}

```

```
// fields/myMultiselectionField.dart *****
```

```
class MyMultiSelectionField<T> extends StatelessWidget {  
  MyMultiSelectionField({  
    Key key,  
    this.values,  
    @required this.options,  
    this.titleBuilder,  
    @required this.chipLabelBuilder,  
    this.hint,  
    @required this.onChanged,  
  }) : assert(options == null ||  
    options.isEmpty ||  
    values == null ||  
    values.every((value) =>  
      options.where((T option) {  
        return option == value;  
      }).length ==  
        1)),  
    assert(chipLabelBuilder != null),  
    assert(onChanged != null),  
    super(key: key);  
  final ValueChanged<List> onChanged;  
  final List<T> values;  
  final List<T> options;  
  final Widget hint;  
  final Widget Function(T) titleBuilder;  
  final Widget Function(T) chipLabelBuilder;  
  @override  
  Widget build(BuildContext context) {  
    return Column(  
      mainAxisAlignment: MainAxisAlignment.spaceBetween,  
      crossAxisAlignment: CrossAxisAlignment.stretch,  
      children: [  
        DropdownButtonHideUnderline(  
          child: DropdownButton<T>(  
            value: null,  
            items: options  
              .map<DropdownMenuItem>(  
                (T option) => DropdownMenuItem(  
                  value: option,  
                  child: MyCheckboxListTile(  
                    selected: values.contains(option),  
                    title: titleBuilder(option),  
                    onChanged: (_) {  
                      if (!values.contains(option)) {  
                        values.add(option);  
                      } else {  
                        values.remove(option);  
                      }  
                      onChanged(values);  
                    },  
                  ),  
                ),  
              ),  
        ],  
      ),  
    ),  
  ),  
}
```

```

        ),
    )
    .toList(),
    selectedItemBuilder: (BuildContext context) {
      return options.map<Widget>((T option) {
        return Text("");
      }).toList();
    }, // Selected items won't be displayed here as they are already displayed in the chip
list
    hint: hint, onChanged: onChanged == null ? null : (T value) {},
  ),
),
  SizedBox(height: 8.0),
  Row(
    children: [
      Expanded(
        child: MyChipList(
          values: values,
          chipBuilder: (T value) {
            return Chip(
              label: chipLabelBuilder(value),
              onDelete: () {
                values.remove(value);
                onChanged(values);
              },
            );
          },
        ),
      ),
    ],
  ),
),
],
),
],
);
}
}

```

```

class MyCheckboxListTile extends StatefulWidget {
  MyCheckboxListTile({
    Key key,
    @required this.title,
    @required this.onChanged,
    @required this.selected,
  }) : assert(title != null),
      assert(onChanged != null),
      assert(selected != null),
      super(key: key);
  final Widget title;
  final dynamic onChanged;
  final bool selected;
  @override
  _MyCheckboxListTileState createState() => _MyCheckboxListTileState();
}

```

```

class _MyCheckboxListTileState extends State<MyCheckboxListTile> {
  _MyCheckboxListTileState();
  bool _checked;
  @override
  void initState() {
    _checked = widget.selected;
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return CheckboxListTile(
      value: _checked,
      selected: _checked,
      title: widget.title,
      controlAffinity: ListTileControlAffinity.leading,
      onChanged: (checked) {
        widget.onChanged(checked);
        setState(() {
          _checked = checked;
        });
      },
    );
  }
}

```

```

class MyChipList<T> extends StatelessWidget {
  const MyChipList({
    @required this.values,
    @required this.chipBuilder,
  });
  final List<T> values;
  final Chip Function(T) chipBuilder;
  List _buildChipList() {
    final List items = [];
    for (T value in values) {
      items.add(chipBuilder(value));
    }
    return items;
  }

  @override
  Widget build(BuildContext context) {
    return Wrap(
      children: _buildChipList(),
    );
  }
}

```

// main.dart

// ...

```

enum Interest {
  Sports,

```

```

Tech,
Games,
Mentoring,
Art,
Travel,
Music,
Reading,
Cooking,
 Blogging
}

```

```

class SignupUser {

```

```

  // ...
  List<Interest> interests;
  SignupUser({

    // ...
    List<Interest> interests,
  }) {
    this.interests = interests ?? [];
  }

```

```

  Map<String, dynamic> toJson() => {

```

```

    // ...
    'interests': interests.toString(),
  };
}

```

```

// ...
class _MyHomePageState extends State {

```

```

  // ...
  @override
  Widget build(BuildContext context) {
    return Scaffold(

      // ...
      body: SafeArea(

        // ...
        child: Form(
          key: _formKey,
          autovalidate: true,
          child: ListView(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            children: [

              // ...
              SizedBox(height: 8.0),
              MyMultiSelectionFormField<Interest>(
                decoration: InputDecoration(
                  labelText: 'Interests',

```

```

    ),
    hint: Text('Select more interests'),
    isDense: true,
    options: Interest.values,
    titleBuilder: (interest) => Text(describeEnum(interest)),
    chipLabelBuilder: (interest) => Text(describeEnum(interest)),
    initialValues: _formResult.interests,
    validator: (interests) => interests.length < 3
      ? 'Please select at least 3 interests'
      : null,
    onSave: (interests) {
      _formResult.interests = interests;
    },
  ),
],
),
),
),
),

// ...
);
}

// ...
}

```

DateFormField

A custom DateFormField for the user birthday, not based on a date picker as I find it heavy when what you only want is the user to enter one fixed date far in the past. As you can see below, this field is composed of 3 TextField input Widgets, validated and saved all at the same time!

// myDateFormField.dart

```

class MyDateFormField extends FormField<DateTime> {
  MyDateFormField({
    Key key,

    DateTime initialValue, // Initial date

    this.dayFocusNode, // FocusNode for the day TextField

    double dayWidth = 40, // Width of the day TextField

    this.monthFocusNode, // FocusNode for the month TextField

    double monthWidth = 40, // Width of the month TextField

    this.yearFocusNode, // FocusNode for the year TextField

    double yearWidth = 60, // Width of the year TextField

    MainAxisAlignment mainAxisAlignment = MainAxisAlignment.start, // Aligment of the

```

TextField widgets

```
InputDecoration inputDecoration = const InputDecoration(  
  border: InputBorder.none, contentPadding: EdgeInsets.all(0)  
),
```

```
Widget separator = const Text('/'), // Widget to place between TextField widgets  
this.onChangeed,
```

```
GestureTapCallback onTap, // Method called when one of the TextField is tapped
```

```
VoidCallback onEditingComplete, // Method called when the last TextField is completed  
FormFieldSetter onSaved,
```

```
FormFieldValidator validator,
```

```
}) : assert(initialValue == null),
```

```
    assert(separator != null),
```

```
    super(  
      key: key,
```

```
      key: key,
```

```
      initialValue: initialValue,
```

```
      onSaved: onSaved,
```

```
      validator: validator,
```

```
      builder: (FormFieldState field) {
```

```
        final _MyDateFormFieldState state = field;
```

```
        final InputDecoration effectiveDecoration = (inputDecoration ??  
          const InputDecoration())  
          .applyDefaults(Theme.of(field.context).inputDecorationTheme);
```

```
        String toOriginalFormatString(DateTime dateTime) {
```

```
          final y = dateTime.year.toString().padLeft(4, '0');
```

```
          final m = dateTime.month.toString().padLeft(2, '0');
```

```
          final d = dateTime.day.toString().padLeft(2, '0');
```

```
          return "$y$m$d";
```

```
        }
```

```
        bool isValidDate(String input) {
```

```
          try {
```

```
            final date = DateTime.parse(input);
```

```
            final originalFormatString = toOriginalFormatString(date);
```

```
            return input == originalFormatString;
```

```
          } catch (e) {
```

```
            return false;
```

```
          }
```

```
        }
```

```
      return InputDecorator(  
        decoration:
```

```
        decoration:
```

```
          effectiveDecoration.copyWith(errorText: field.errorText),
```

```
        isEmpty: false,
```

```
        isFocused: state._effectiveYearFocusNode.hasFocus ||
```

```
          state._effectiveMonthFocusNode.hasFocus ||
```

```
          state._effectiveDayFocusNode.hasFocus,
```

```
        child: Row(mainAxisAlignment: mainAxisAlignment, children: [
```

```
          SizedBox(  
            width: dayWidth,
```

```
            width: dayWidth,
```

```
            child: TextField(  
              controller: state._effectiveDayController,
```

```
              controller: state._effectiveDayController,
```



```

inputFormatters: [
  LengthLimitingTextInputFormatter(2),
  WhitelistingTextInputFormatter.digitsOnly,
],
decoration: inputDecoration,
focusNode: state._effectiveDayFocusNode,
keyboardType: TextInputType.number,
onChanged: (value) {
  if (value.length == 2 &&
    int.parse(value) > 0 &&
    int.parse(value) <= 31) {
    state._effectiveMonthFocusNode.requestFocus();
  }
  if (value != " &&
    state._effectiveMonthController.text != " &&
    state._effectiveYearController.text != ") {
    final date =
'${state._effectiveYearController.text}${state._effectiveMonthController.text}$value';
    if (isValidDate(date)) {
      field.didChange(DateTime.utc(
        int.parse(state._effectiveYearController.text),
        int.parse(state._effectiveMonthController.text),
        int.parse(value),
      ));
    } else {
      field.didChange(null);
    }
  }
},
onEditingComplete: () {
  state._effectiveMonthFocusNode.requestFocus();
},
),
),
separator,
SizedBox(
width: monthWidth,
child: TextField(
  controller: state._effectiveMonthController,
  inputFormatters: [
    LengthLimitingTextInputFormatter(2),
    WhitelistingTextInputFormatter.digitsOnly,
  ],
  decoration: inputDecoration,
  focusNode: state._effectiveMonthFocusNode,
  keyboardType: TextInputType.number,
  onChanged: (value) {
    if (value.length == 2 &&
      int.parse(value) > 0 &&
      int.parse(value) <= 12) {
      state._effectiveYearFocusNode.requestFocus();
    }
    if (value != " &&

```

```

        state._effectiveDayController.text != " &&
        state._effectiveYearController.text != ") {
    final date =

'$ {state._effectiveYearController.text}$value$ {state._effectiveDayController.text}';
    if (isValidDate(date)) {
        field.didChange(DateTime.utc(
            int.parse(state._effectiveYearController.text),
            int.parse(value),
            int.parse(state._effectiveDayController.text),
        ));
    } else {
        field.didChange(null);
    }
}
},
onTap: onTap,
onEditingComplete: () {
    state._effectiveYearFocusNode.requestFocus();
},
),
),
separator,
SizedBox(
    width: yearWidth,
    child: TextField(
        controller: state._effectiveYearController,
        inputFormatters: [
            LengthLimitingTextInputFormatter(4),
            WhitelistingTextInputFormatter.digitsOnly,
        ],
        decoration: inputDecoration,
        focusNode: state._effectiveYearFocusNode,
        keyboardType: TextInputType.number,
        onChanged: (value) {
            if (value != " &&
                state._effectiveDayController.text != " &&
                state._effectiveMonthController.text != ") {
                final date =

'$value$ {state._effectiveMonthController.text}$ {state._effectiveDayController.text}';
                if (isValidDate(date)) {
                    field.didChange(DateTime.utc(
                        int.parse(value),
                        int.parse(state._effectiveMonthController.text),
                        int.parse(state._effectiveDayController.text),
                    ));
                } else {
                    field.didChange(null);
                }
            }
        },
        onTap: onTap,
        onEditingComplete: onEditingComplete,

```

```

        ),
    ),
  ]));
},
);
final ValueChanged onChanged;
final FocusNode dayFocusNode;
final FocusNode monthFocusNode;
final FocusNode yearFocusNode;
@override
_MyDateFormFieldState createState() => _MyDateFormFieldState();
}

class _MyDateFormFieldState extends FormFieldState<DateTime> {
  @override
  MyDateFormField get widget => super.widget;
  TextEditingController _dayController;
  TextEditingController get _effectiveDayController => _dayController;
  TextEditingController _monthController;
  TextEditingController get _effectiveMonthController => _monthController;
  TextEditingController _yearController;
  TextEditingController get _effectiveYearController => _yearController;
  FocusNode _dayFocusNode;
  FocusNode get _effectiveDayFocusNode => widget.dayFocusNode ?? _dayFocusNode;
  FocusNode _monthFocusNode;
  FocusNode get _effectiveMonthFocusNode =>
    widget.monthFocusNode ?? _monthFocusNode;
  FocusNode _yearFocusNode;
  FocusNode get _effectiveYearFocusNode =>
    widget.yearFocusNode ?? _yearFocusNode;
  @override
  void initState() {
    super.initState();
    _dayController = TextEditingController(
      text: widget.initialValue != null
        ? widget.initialValue.day.toString()
        : "");
    _monthController = TextEditingController(
      text: widget.initialValue != null
        ? widget.initialValue.month.toString()
        : "");
    _yearController = TextEditingController(
      text: widget.initialValue != null
        ? widget.initialValue.year.toString()
        : "");
    if (widget.dayFocusNode == null) {
      _dayFocusNode = FocusNode();
    }
    if (widget.monthFocusNode == null) {
      _monthFocusNode = FocusNode();
    }
    if (widget.yearFocusNode == null) {
      _yearFocusNode = FocusNode();
    }
  }
}

```

```

}

@override
void didChange(DateTime value) {
  super.didChange(value);
  if (widget.onChanged != null) {
    widget.onChanged(value);
  }
}

@override
void reset() {
  super.reset();
  setState(() {
    _effectiveDayController.text = widget.initialValue != null
      ? widget.initialValue.day.toString()
      : null;
    _effectiveMonthController.text = widget.initialValue != null
      ? widget.initialValue.month.toString()
      : null;
    _effectiveYearController.text = widget.initialValue != null
      ? widget.initialValue.year.toString()
      : null;
  });
}
}

```

// main.dart

// ...

```

class SignupUser {

  // ...
  DateTime birthdate;
  SignupUser({

    // ...
    this.birthdate,
  });
  Map<String, dynamic> toJson() => {

    // ...
    'birthdate': birthdate.toString(),
  };
}

// ...
class _MyHomePageState extends State {

  // ...
  @override
  Widget build(BuildContext context) {
    return Scaffold(

```

```

// ...
body: SafeArea(

  // ...
  child: Form(
    key: _formKey,
    autovalidate: true,
    child: ListView(
      padding: const EdgeInsets.symmetric(horizontal: 16.0),
      children: [

        // ...
        SizedBox(height: 8.0),
        MyMultiSelectionFormField<Interest>(
          decoration: InputDecoration(
            labelText: 'Interests',
          ),
          hint: Text('Select more interests'),
          isDense: true,
          options: Interest.values,
          titleBuilder: (interest) => Text(describeEnum(interest)),
          chipLabelBuilder: (interest) => Text(describeEnum(interest)),
          initialValues: _formResult.interests,
          validator: (interests) => interests.length < 3
            ? 'Please select at least 3 interests'
            : null,
          onSave: (interests) {
            _formResult.interests = interests;
          },
        ),
      ],
    ),
  ),
);
}

// ...
}

```

In addition I added one FocusNode for each text input in order to move the focus from one input field to the other when edition is completed.

That's it, we have a complete signup form now, and it's easy to submit all the information the user entered with the onSave method at the form level.

Signup Form

DEBUG

Name
Jo

Gender

Male Female Other

☐ Sports

☐ Tech

☒ Games

☐ Mentoring

☐ Art

☐ Travel

☐ Music

☐ Reading

☐ Cooking

☐ Blogging

✓

Focusing the right field

Another point is to improve the UX by handling properly which field to focus on. FocusNode class is what we need to handle focus. First add the `focusNode(s)` property to the previous fields. Then define one `focusNode` for each input in your form (some fields have multiple inputs such as Toggle or Date) and pass it to your fields:

```

// main.dart

// ...

// ...
class _MyHomePageState extends State {
  final GlobalKey _formKey = GlobalKey();
  final _formResult = SignupUser();

  final nameFocusNode = FocusNode();
  final genderFocusNodes = [FocusNode(), FocusNode(), FocusNode()];
  final birthdateFocusNodes = [FocusNode(), FocusNode(), FocusNode()];
  final interestsFocusNode = FocusNode();
  final ethicsAgreementFocusNode = FocusNode();

  @override
  Widget build(BuildContext context) {
    return Scaffold(

      // ...
      body: SafeArea(

        // ...
        child: Form(
          key: _formKey,
          autovalidate: true,
          child: ListView(
            padding: const EdgeInsets.symmetric(horizontal: 16.0),
            children: [
              TextFormField(
                focusNode: nameFocusNode,

                // ...
              ),
              MyToggleButtonsFormField<Gender>(
                focusNodes: genderFocusNodes,

                // ...
              ),
              // etc.
            ],
          ),
        ),
      ),
    );
  }

  // ...
}

```

Then set the 'isFocused' parameter of each InputDecorator with help of the 'focusNode.hasFocus' property.

For example:

```
// main.dart

// ...

return InputDecoration(
  decoration: effectiveDecoration.copyWith(errorText: field.errorText),
  isEmpty: false,

  // Fields with one FocusNode (ex: MySwitchFormField)
  isFocused: focusNode.hasFocus,

  // OR

  // Fields with multiple FocusNode (ex: MyToggleButtonsFormField)
  isFocused: focusNodes?.any((focusNode) => focusNode.hasFocus),

  // ...
);

// ...
```

And to conclude, at the form level, request focus for the associated node in the `onChange` or `onTap` method to assure that it will focus on the field when the user taps on it.


```

// main.dart

// ...

// Fields with TextField inside (ex: TextFormField)
TextFormField(
  onTap: () {
    FocusScope.of(context).unfocus();
    FocusScope.of(context).requestFocus(nameFocusNode);
  },

  // ...
),

// Other fields (ex: MyToggleButtonsFormField)
MyToggleButtonsFormField<Gender>(
  onChanged: (gender) {
    final genderIndex = Gender.values.indexOf(gender);
    if (genderIndex >= 0) {
      FocusScope.of(context).unfocus();
      FocusScope.of(context).requestFocus(
        genderFocusNodes[genderIndex]
      );
    }
  },

  // ...
),

// ...

```

Validating when necessary

I am not a big fan of autovalidation on the form as it makes a lot of error messages appear when the user starts answering an empty form. A solution is to set the `autovalidate` attribute to `false` both on form and fields, and to only call `form.validate()` at the form submission. In addition you can modify the internal `onChanged` behavior of each field in order to call `field.validate()` when the user update the value and the field has an error.

For example:

```
// myMultiSelectionFormField.dart

// ...

class _MyMultiSelectionFormFieldState<T> extends FormFieldState<List<T>> {
  @override
  MyMultiSelectionFormField<T> get widget => super.widget;
  @override
  void didChange(List values) {
    super.didChange(values);
    if (this.hasError) {
      this.validate();
    }
    if (widget.onChanged != null) {
      widget.onChanged(values);
    }
  }
}
```

If you give a look in the DartPad you can notice that I've used a custom implementation of TextFormField just for this feature.

That's all 😊

If you have any questions/feedbacks or if you know other ways to do it, feel free to post a comment or send a Tweet at [@AntoineFrancon](#). On the other hand feel free to use or extend the few widgets I presented in this article for building your own FormField library!