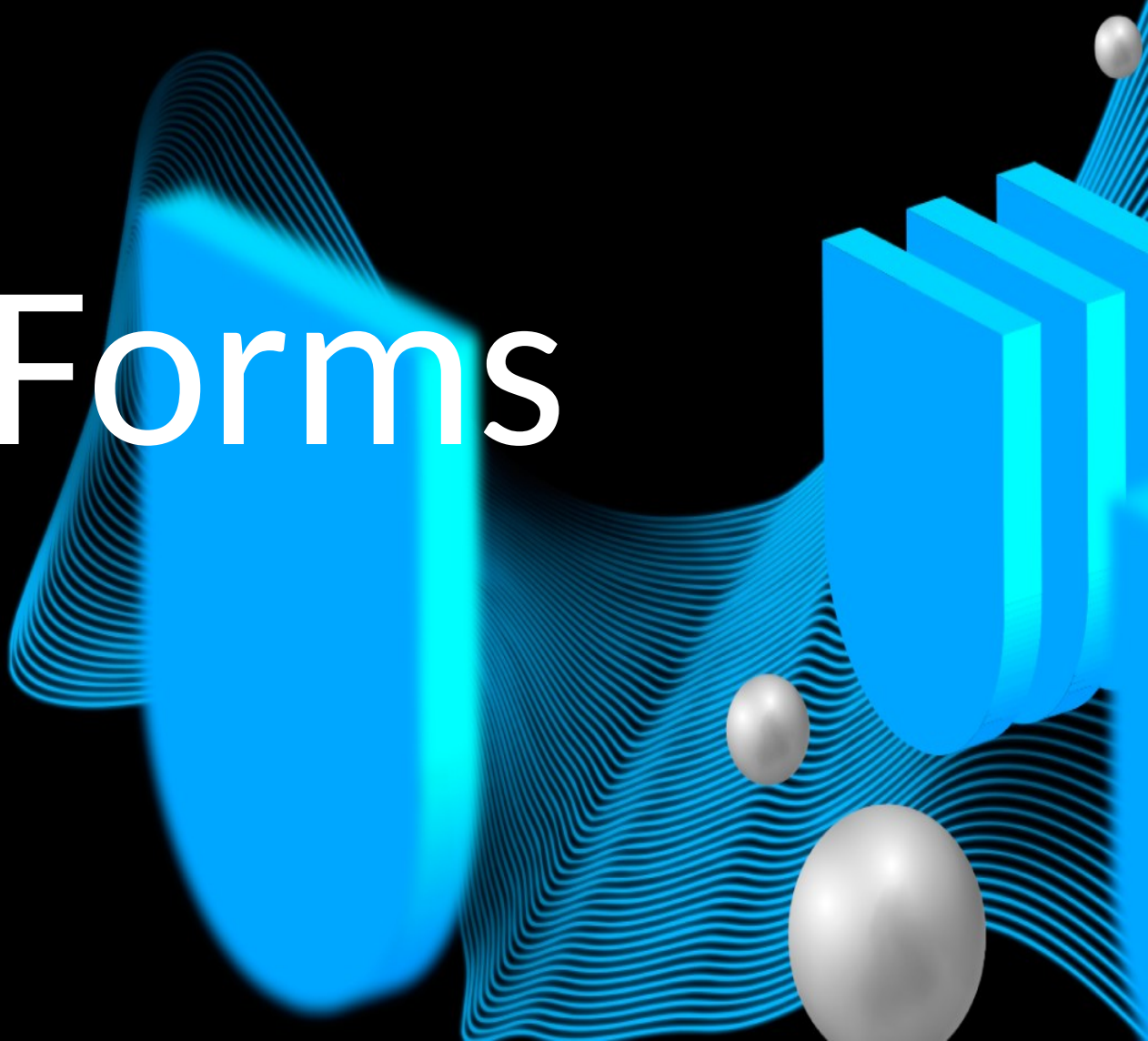

UNITAU
Universidade de Taubaté

Forms



Validação

Para criar telas de entrada de dados poder ser utilizada a classe **Form** que provê mecanismos para gerenciamento dos widgets com os dados.

O formulário utiliza o **FormState** para validar, resetar e salvar os widgets do formulário.

Validação

Crie uma nova aplicação Flutter com o nome validation e altere main.dart para:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Form Validation',
      theme: ThemeData(primarySwatch: Colors.blue,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: Home(),
    );
  }
}
```

Validação

Crie o subdiretório **lib/pages** e o arquivo **home.dart** nesse subdiretório.

Inclua a classe para os dados do pedido:

```
import 'package:flutter/material.dart';
//
// Classe para os dados do pedido
//
class Order {
  String item;
  int quantity;
}
```

Acrescente a classe com o formulário dos dados:

```
//
// Classe para o formulario dos dados
//
class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}
```

Validação

Acrescente a classe para gerenciar o estado do formulário:

```
//
// Classe para gerenciar o estado do formulario
//
class _HomeState extends State<Home> {
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  // Chave para o Scaffold do formulario
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
  // Dados do pedido
  Order _order = Order();
```

Validação

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _scaffoldKey,
      appBar: AppBar(
        title: Text('Form Validation'),
      ),
      body: SafeArea(
        child: Column(
          children: <Widget>[
          ],
        ),
      ),
    );
  }
}
```

Validação

Temos a tela principal com um objeto que será a chave do formulário. Esta chave será usada para acessar os métodos de gerenciamento do formulário:

```
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
```

Temos um objeto para chave do Scaffold do formulário. Essa chave é utilizada para acessar elementos do Scaffold:

```
  // Chave para o Scaffold do formulario
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
```

E um objeto para modelo de dados que será utilizado para os valores iniciais do formulário. Após a validação da entrada de dados, os novos dados serão armazenados nesse mesmo objeto:

```
  // Dados do pedido
  Order _order = Order();
```

Validação

Acrescente o widget para o formulário dentro da SafeArea:

```
children: <Widget>[
  Form(
    key: _formStateKey,
    autovalidate: true,
    child: Padding(
      padding: EdgeInsets.all(16.0),
      child: Column(
        children: <Widget>[
          TextFormField(
            decoration: InputDecoration(
              hintText: 'Nome',
              labelText: 'Item',
            ),
            validator: (value) =>
_validateItem(value),
            onSaved: (value) {
              print("Salvando item");
              _order.item = value;
            },
          ),
```

Validação

```
                    TextFormField(
                      decoration:  InputDecoration(
                        hintText: '1',
                        labelText: 'Quantidade',
                      ),
                      validator: (value) =>
_validateQuantity(value),
                      onSaved: (value) {
                        print("Salvando quantidade");
                        _order.quantity = int.parse(value);
                      }
                    ),
                  ],
                ),
              ),
            ),
```

A propriedade **autovalidate** determina que os widget do formulário sejam validados imediatamente quando alterados.

Validação

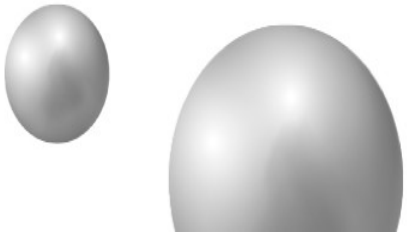
Acrescente os botões para ações do formulário após o body do Scaffold:

```
bottomNavigationBar: ButtonBar(
  alignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    RaisedButton(
      color: Colors.blue[400],
      onPressed: () =>
          _formStateKey.currentState.reset(),
      child: Text('Reset')
    ),
    RaisedButton(
        color: Colors.blue[400],
      onPressed: () => _submitOrder(),
      child: Text('Save')
    ),
  ],
),
```

Validação

Acrescente os métodos para validação dos dados do formulário dentro da classe _HomeState. Esses métodos devem retornar null se o conteúdo do campo é válido, ou uma string indicando o erro caso o conteúdo seja inválido:

```
//
// Valida o nome do item
//
String _validateItem(String value) {
  String ret = null;
  value = value.trim();
  print("Validando item [${value}]");
  if ( value.isEmpty )
    ret = "Nome e obrigatorio";
  return ret;
}
```



Validação

```
//
// Valida a quantidade
//
String _validateQuantity(String value) {
  String ret = null;
  value = value.trim();
  print("Validando quantidade [${value}]");
  if ( value.isEmpty )
    ret = "Quantidade e obrigatoria";
  else {
    int _valueAsInteger = int.tryParse(value);
    if ( _valueAsInteger == null )
      ret = "Valor invalido";
    else {
      if ( _valueAsInteger <= 0 )
        ret = "Valor deve ser maior que zero";
    }
  }
  return ret;
}
```

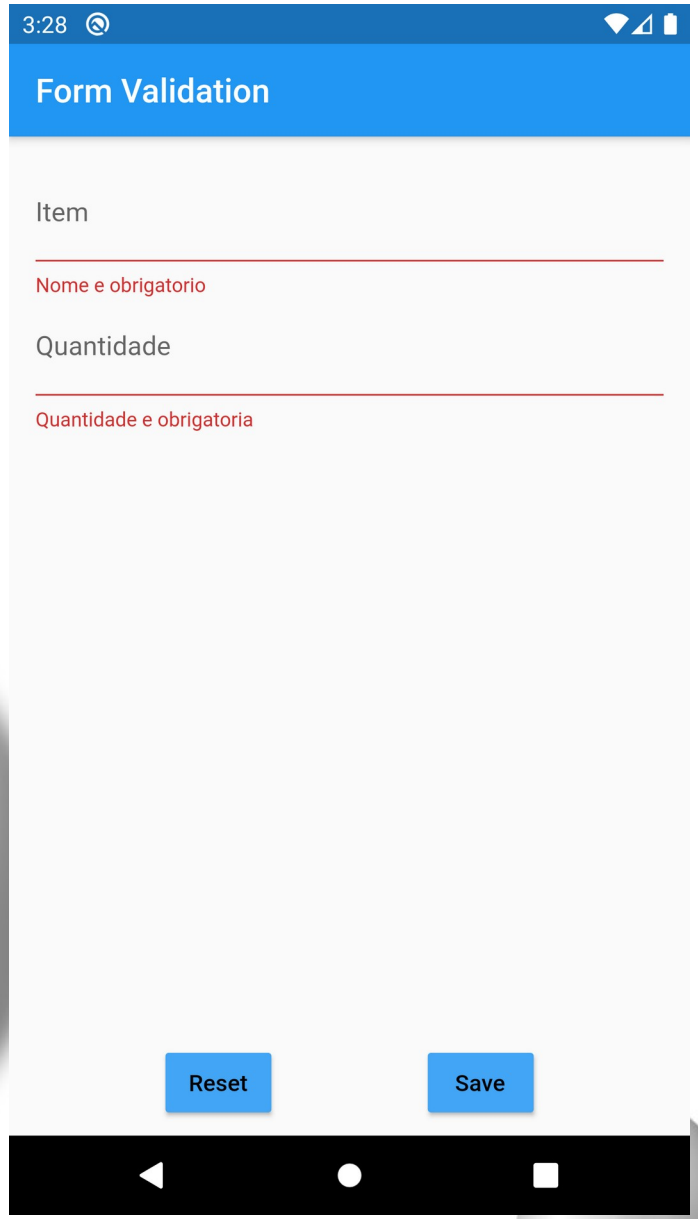
Validação

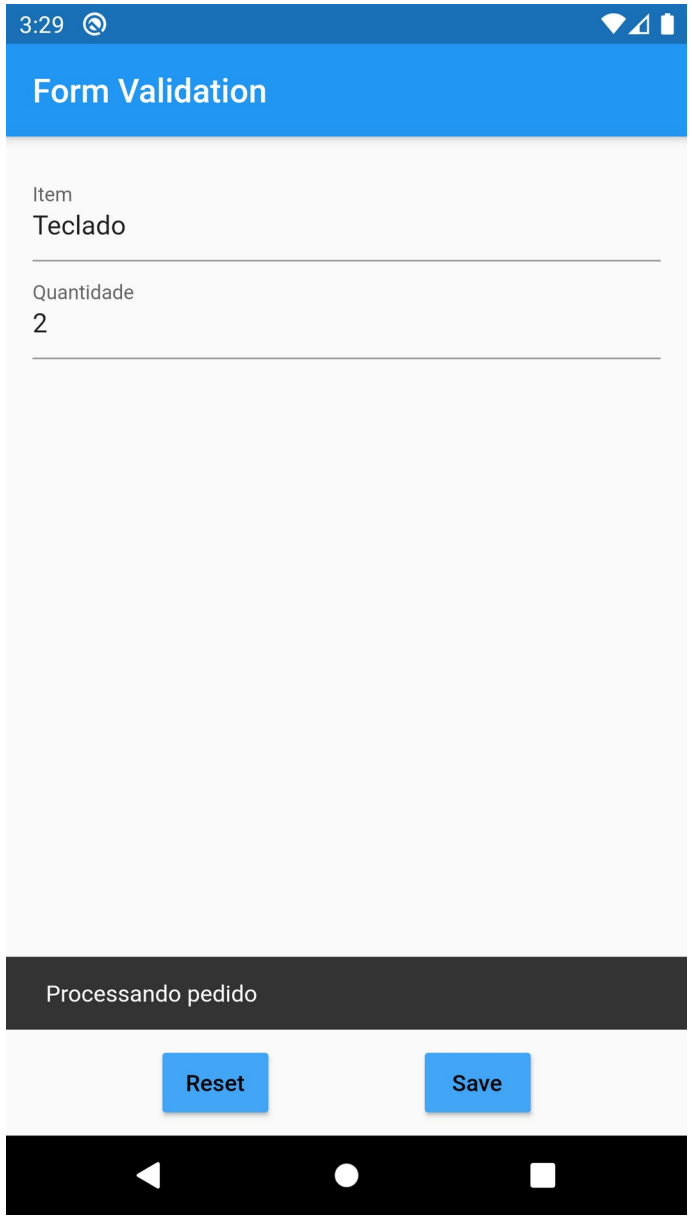
Acrescente o método para verificar e salvar os dados do formulário dentro da classe _HomeState:

```
//
// Verifica e salva os dados do formulario
//
_submitOrder() {
  if(_formStateKey.currentState.validate()) {
    _formStateKey.currentState.save();
    print('Order Item: ${_order.item}');
    print('Order Quantity: ${_order.quantity}');
    _scaffoldKey.currentState.showSnackBar(
       SnackBar(content: Text('Processando pedido'))
    );
  }
}
```

O método **validate** do **FormState** irá executar a função de validação de cada um widgets do formulário (**validator**) e o método **save** irá executar a função para salvar os dados (**onSave**) de cada um dos widgets do formulário.

Validação


3:28
Form Validation
Item
Nome e obrigatorio
Quantidade
Quantidade e obrigatoria
Reset
Save


3:29
Form Validation
Item
Teclado
Quantidade
2
Processando pedido
Reset
Save

Widgets

Flutter possui vários widgets para entrada de dados como **Checkbox**, **DropdownButton**, **Radio**, **Switch** e **TextField**.

Alguns widgets para entrada de dados não mantém um estado. Para gerenciar o estado e facilitar a atualização e validação dos widgets de entrada de dados, o **FormField** utiliza um objeto **FormFieldState**. Existem algumas classes já definidas para combinar os widgets de entrada de dados com um **FormField** como **DropdownButtonFormField** e **TextFormField**.

Um **FormField** pode ser utilizados sozinho, mas usualmente, é utilizado com a classe **Form**, um contêiner para os widgets de entrada de dados que utiliza um **FormState** para tratar esses widgets como um grupo.

Widgets

Crie uma nova aplicação Flutter com o nome form_widgets e altere main.dart para:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter Demo',
      theme: ThemeData(primarySwatch: Colors.blue,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: Home(),
    );
  }
}
```

Widgets

Crie o subdiretório lib/models e o arquivo user.dart nesse subdiretório:

```
class User {
  // Lista de preferencias
  static const String PreferenciasHardware = 'hardware';
  static const String PreferenciasDesenvolvimento =
'desenvolvimento';
  static const String PreferenciasRedes = 'redes';
  // Lista de areas de atuacao
  final List <String>Atuacao =
["Desenvolvedor","Gestor","Estudante"];
  // Dados do usuario
  String nome = '';
  int idade;
  String estciv;
  String atuacao;
  Map preferencias = {
    PreferenciasHardware: false,
    PreferenciasDesenvolvimento: false,
    PreferenciasRedes: false };
  bool newsletter = false;
```

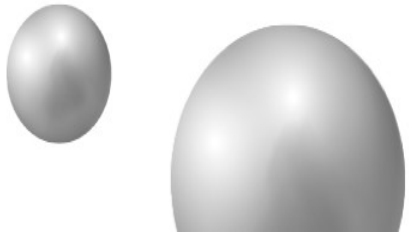
Widgets

```
  // Salvando os dados do modelo
  save() {
    print('Salvando os dados');
  }

  // Exibe os dados do modelo
  show() {
    print("Dados do usuario");
    print("  Nome         : ${this.nome}");
    print("  Idade        : ${this.idade}");
    print("  Estado Civil: ${this.estciv}");
    print("  Atuacao      : ${this.atuacao}");
    print("  NewsLetter   : ${this.newsletter?"Sim":"Nao"}");
    print("  Preferencias:");
    if ( this.preferencias[PreferenciasHardware] )
      print("      Hardware");
    if ( this.preferencias[PreferenciasDesenvolvimento] )
      print("      Desenvolvimento");
    if ( this.preferencias[PreferenciasRedes] )
      print("      Redes");
  }
}
```

Widgets

Crie o subdiretório **lib/pages** e o arquivo **home.dart** nesse subdiretório. Acrescente a classe com o formulário dos dados:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import '../models/user.dart';

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}
```



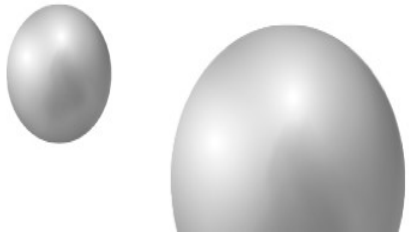
Widgets

Acrescente a classe para gerenciar o estado do formulário. As variáveis desenvolvimento e redes foram criadas para contornar problemas no uso direto do mapa no initialValue do FormField:

```
class _HomeState extends State {
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  // Chave para o Scaffold
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
  // Modelo de dados
  final _user = User();

  @override
  Widget build(BuildContext context) {
    bool desenvolvimento =
_user.preferencias[User.PreferenciasDesenvolvimento];
    bool redes = _user.preferencias[User.PreferenciasRedes];
```

Widgets

```
    return Scaffold(key: _scaffoldKey,
      appBar: AppBar(title: Text('Profile')),
      body: Container(
        padding: const EdgeInsets.symmetric(vertical: 16.0,
horizontal: 16.0),
        child: Builder(
          builder: (context) => Form(
            key: _formStateKey,
            autovalidate: true,
            child: SingleChildScrollView(
              child: Column(
                crossAxisAlignment:
CrossAxisAlignment.stretch,
                children: <Widget>[
                ]
              ),
            )
          )
        )
      ),
    );
  }
```



Widgets

Acrescente os widgets para os dados do formulário dentro do Form.

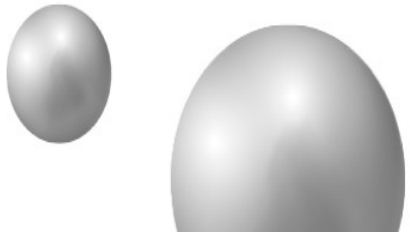
```
TextFormField(
  initialValue: _user.nome,
  decoration: InputDecoration(labelText: 'Nome'),
  validator: (value) => _validateNome(value),
  onSaved: (value) => setState(() => _user.nome = value),
),
TextFormField(
  initialValue:
    (_user.idade ?? 0)>=18 ? _user.idade.toString() : '',
  InputFormatters:
    [WhitelistingTextInputFormatter.digitsOnly],
  keyboardType: TextInputType.number,
  decoration: InputDecoration(labelText: 'Idade'),
  validator: (value) => _validateIdade(value),
  onSaved: (value) =>
    setState(() => _user.idade = int.tryParse(value)),
),
Divider(),
```

Widgets

```
FormField(
  validator: (value) => _validateEstCiv(value),
  onSaved: (value) =>
    setState(() => _user.estciv = value),
  initialValue: _user.estciv,
  builder: (FormFieldState<String> state) {
    return Column(children: <Widget>[
      ListTile(
        title: Text('Estado Civil:'),
      ),
      RadioListTile<String>(
        title: const Text('Solteiro'),
        value: 'solteiro',
        groupValue: state.value,
        onChanged: (value) => state.didChange(value),
      ),
      RadioListTile<String>(
        title: const Text('Casado'),
        value: 'casado',
        groupValue: state.value,
        onChanged: (value) => state.didChange(value),
      ),
```

Widgets

```
        RadioListTile<String>(
          title: const Text('Outro'),
          value: 'outro',
          groupValue: state.value,
          onChanged: (value) => state.didChange(value),
        ),
        state.hasError ?
          Text(
            state.errorText,
            style: TextStyle(color: Colors.red),
          )
        :
          Container()
      ]);
    }
),
Divider(),
```



Widgets

```
FormField(
  validator: (value) => _validateAtuacao(value),
  onSaved: (value) =>
    setState(() => _user.atuacao = value),
  initialValue: _user.atuacao,
  builder: (FormFieldState<String> state) {
    return Column(children: <Widget>[
      ListTile(
        title: Text('Atuacao'),
        trailing: DropdownButton<String>(
          value: state.value,
          icon: Icon(Icons.arrow_downward),
          iconSize: 24,
          elevation: 16,
          onChanged: (value) => state.didChange(value),
```

Widgets

```
            items: _user.Atuacao
              .map<DropdownMenuItem<String>>((String value)
{
                return DropdownMenuItem<String>(
                  value: value,
                  child: Text(value),
                );
              }
            ).toList(),
          ),
        ),
        state.hasError ?
          Text(
            state.errorText,
            style: TextStyle(color: Colors.red),
          )
        :
          Container()
      ]);
    }
  ),
  Divider(),
```

Widgets

```
FormField(
  onSaved: (value) =>
    setState(() => _user.newsletter = value),
  initialValue: _user.newsletter?? false,
  builder: (FormFieldState<bool> state) {
    return SwitchListTile(
      title: const Text('Enviar Newsletter'),
      value: state.value,
      onChanged: (bool value) => state.didChange(value)
    );
  }
),
Divider(),
```

Widgets

```
  CheckboxListTile(
    title: const Text('Hardware'),
    value: _user.preferencias[User.PreferenciasHardware],
    onChanged: (val) {
      SetState(() =>
        _user.preferencias[User.PreferenciasHardware] =
val);
    }
  ),
```

Widgets

```
  FormField(
    onSaved: (value) =>
      SetState(() =>
       _user.preferencias[User.PreferenciasDesenvolvimento]
= value
    ),
    initialValue: desenvolvimento?? false,
    builder: (FormFieldState<bool> state) {
      return CheckboxListTile(
        title: const Text('Desenvolvimento'),
        value: state.value,
        onChanged: (bool value) => state.didChange(value)
      );
    }
  ),
```

Widgets

```
FormField(
  onSaved: (value) =>
    setState(() =>
      _user.preferencias[User.PreferenciasRedes] = value
  ),
  initialValue: redes?? false,
  builder: (FormFieldState<bool> state) {
    return CheckboxListTile(
      title: const Text('Redes'),
      value: state.value,
      onChanged: (bool value) => state.didChange(value)
    );
  }
),
```

Widgets

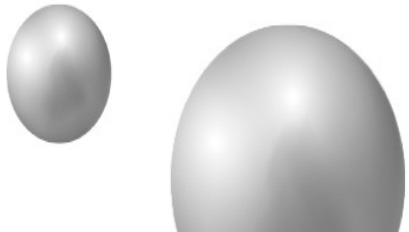
Acrescente os botões para ações do formulário após o body do Scaffold:

```
bottomNavigationBar: ButtonBar(
  alignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    RaisedButton(
        onPressed: () => _user.show(),
        child: Text('Exibir')
    ),
    RaisedButton(
        onPressed: () => _submit(),
        child: Text('Save')
    ),
  ],
),
```

Widgets

Acrescente os métodos para validação dos dados do formulário dentro da classe _HomeState.

```
//
// Validar o nome
//
String _validateNome(String value) {
  String ret = null;
  value = value.trim();
  if ( value.isEmpty )
    ret = "Enter com o nome";
  return ret;
}
```



Widgets

```
//
// Validar a idade
//
String _validateIdade(String value) {
  String ret = null;
  value = value.trim();
  if ( value.isEmpty )
    ret = "Idade e obrigatoria";
  else {
    int _valueAsInteger = int.tryParse(value);
    if ( _valueAsInteger == null )
      ret = "Valor invalido";
    else {
      if ( _valueAsInteger < 18 )
        ret = "Valor deve ser maior que 18";
    }
  }
  return ret;
}
```

Widgets

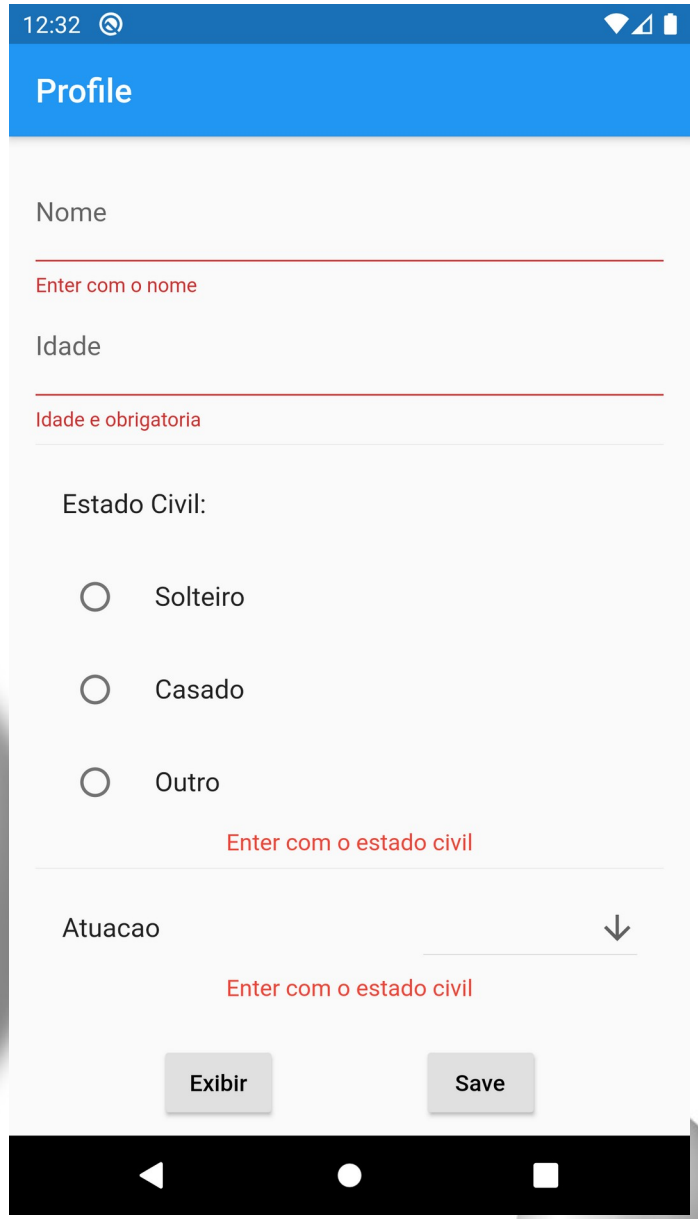
```
//
// Validar o estado civil
//
String _validateEstCiv(String value) {
  String ret = null;
  if ( value == null )
    ret = "Enter com o estado civil";
  return ret;
}

//
// Validar a atuacao
//
String _validateAtuacao(String value) {
  String ret = null;
  if ( value == null )
    ret = "Enter com o estado civil";
  return ret;
}
```

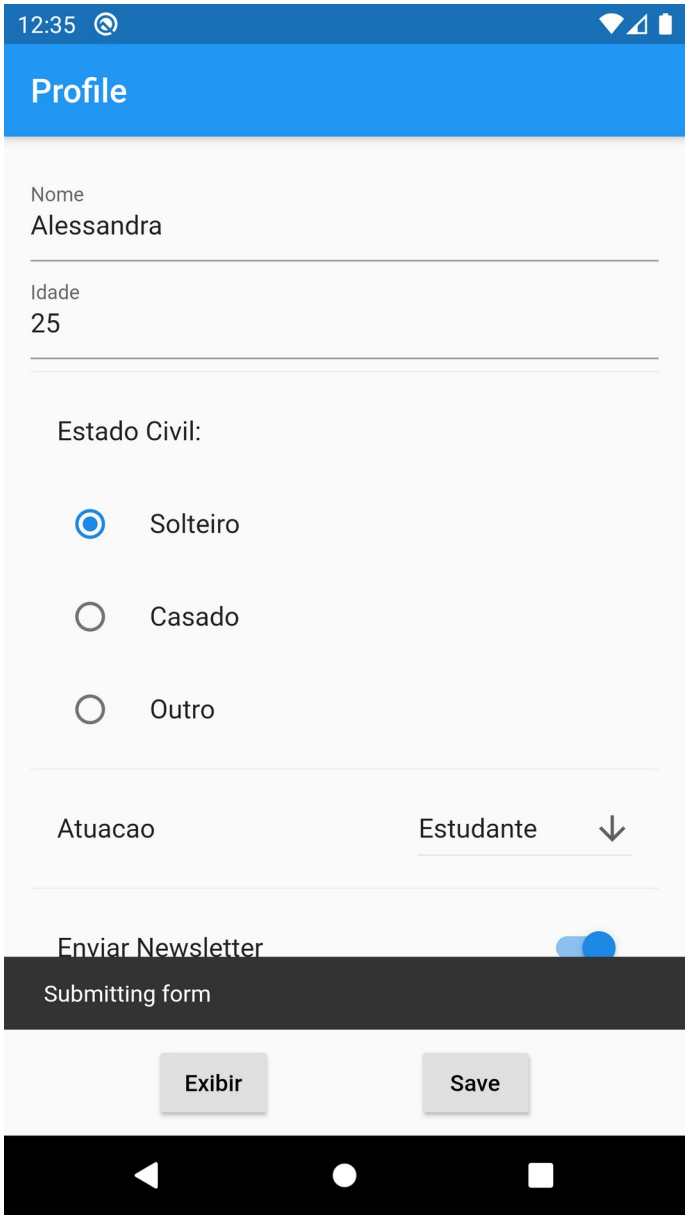
Widgets

Acrescente o método para verificar e salvar os dados do formulário dentro da classe _HomeState:

```
//
// Verifica e salva os dados do formulario
//
_submit() {
  if(_formStateKey.currentState.validate()) {
    _formStateKey.currentState.save();
    _user.save();
    _user.show();
    _scaffoldKey.currentState.showSnackBar(
        SnackBar(content: Text('Submitting form'))
    );
  }
}
```

Widgets


12:32
Profile
Nome
Enter com o nome
Idade
Idade e obrigatoria
Estado Civil:
Solteiro
Casado
Outro
Enter com o estado civil
Atuacao
Enter com o estado civil
Exibir
Save


12:35
Profile
Nome
Alessandra
Idade
25
Estado Civil:
Solteiro
Casado
Outro
Atuacao
Estudante
Enviar Newsletter
Submitting form
Exibir
Save

Focus

Para gerenciar o foco dos widgets, devemos associar ser usado um objeto **FocusNode** a cada widget que se deseja atribuir o foco e controlar os eventos do teclado e um objeto **FocusScopeNode** definir o escopo do controle de foco.

Focus

Crie uma nova aplicação Flutter com o nome form_focus. Crie os mesmos diretórios criados para a aplicação anterior e copie os arquivos dentro desses diretórios.

Acrescente as chaves para os widgets do nome e idade dentro da classe _HomeState:

```
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  // Chave para o Scaffold
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
  // Chave para o widget do nome
  final GlobalKey<FormFieldState> _nomeKey =
GlobalKey<FormFieldState>();
  // Chave para o widget da idade
  final GlobalKey<FormFieldState> _idadeKey =
GlobalKey<FormFieldState>();
```

Focus

Acrescente o objeto para escopo do foco e para foco dos widgets do nome e idade dentro da classe _HomeState:

```
// Modelo de dados
final _user = User();
// Scopo de foco para a aplicacao
FocusScopeNode _focusScopeNode;
// Gerenciar o foco do widget para nome
FocusNode _nomeFocusNode;
// Gerenciar o foco do widget para idade
FocusNode _idadeFocusNode;
```

Focus

Acrescente o método para inicializar o estado na classe _HomeState:

```
@override
void initState(){
  super.initState();
  _nomeFocusNode = FocusNode();
  _idadeFocusNode = FocusNode();
  WidgetsBinding.instance.addPostFrameCallback((_){
    FocusScope.of(context).requestFocus(_nomeFocusNode);
  });
}
```

Focus

Acrescente o método para descartar os objetos da classe _HomeState:

```
@override
void dispose() {
  // Clean up the focus node when the Form is disposed.
  _nomeFocusNode.dispose();
  _idadeFocusNode.dispose();
  super.dispose();
}
```

Focus

Defina o escopo do foco no método build da classe:

```
  @override
  Widget build(BuildContext context) {
    bool desenvolvimento =
_user.preferencias[User.PreferenciasDesenvolvimento];
    bool redes = _user.preferencias[User.PreferenciasRedes];
    _focusScopeNode = FocusScope.of(context);
```

Focus

Acrescente a chave e objeto para gerenciamento do foco nos widgets para nome e idade:

```
TextFormField(
  key: _nomeKey,
  focusNode: _nomeFocusNode,
  initialValue: _user.nome,
  decoration: InputDecoration(labelText:
'Nome'),

TextFormField(
  key: _idadeKey,
  focusNode: _idadeFocusNode,
  initialValue:
    (_user.idade ?? 0)>=18 ?
_user.idade.toString() : '',
```

Focus

Acrescente a variável para o valor inicial do widget para a preferência por hardware:

```
  @override
  Widget build(BuildContext context) {
    bool hardware =
_user.preferencias[User.PreferenciasHardware];
    bool desenvolvimento =
_user.preferencias[User.PreferenciasDesenvolvimento];
    bool redes = _user.preferencias[User.PreferenciasRedes];
```

Focus

Altere o widget para a preferência por hardware:

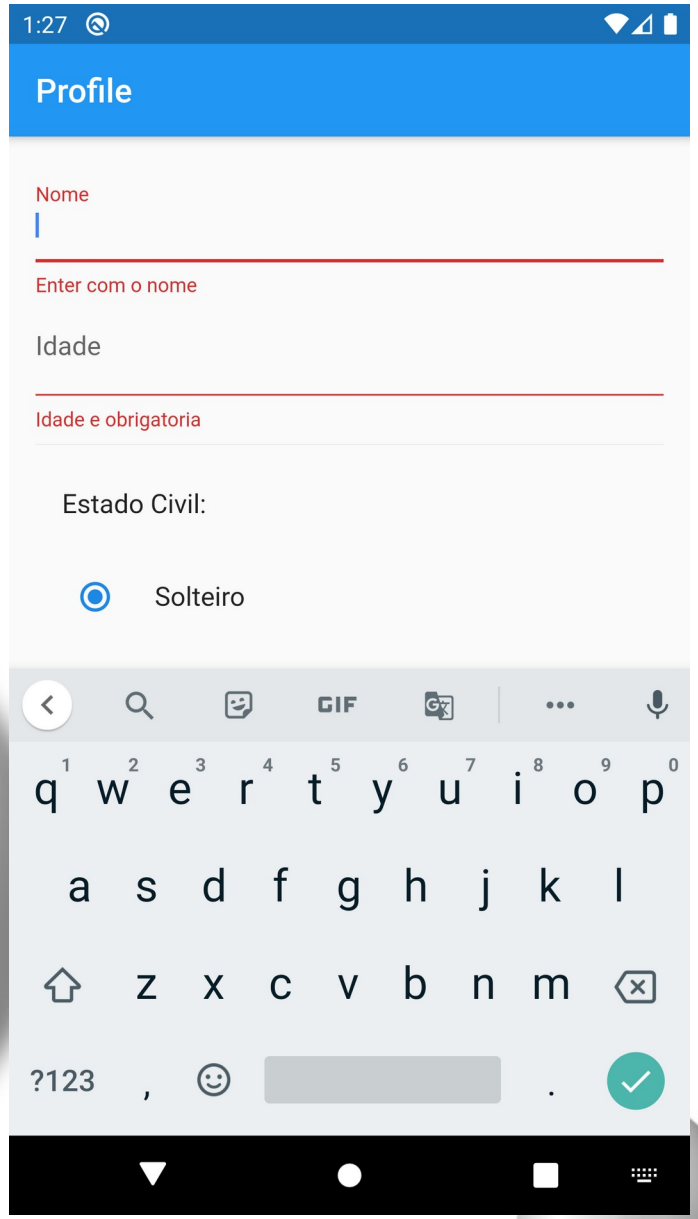
```
                    FormField(
                      onSaved: (value) =>
                        setState(() =>
 _user.preferencias[User.PreferenciasHardware] = value
                      ),
                      initialValue: hardware?? false,
                      builder: (FormFieldState<bool> state) {
                        return CheckboxListTile(
                          title: const Text('Hardware'),
                          value: state.value,
                          onChanged: (bool value) =>
state.didChange(value)
                        );
                      }
                    ),
```

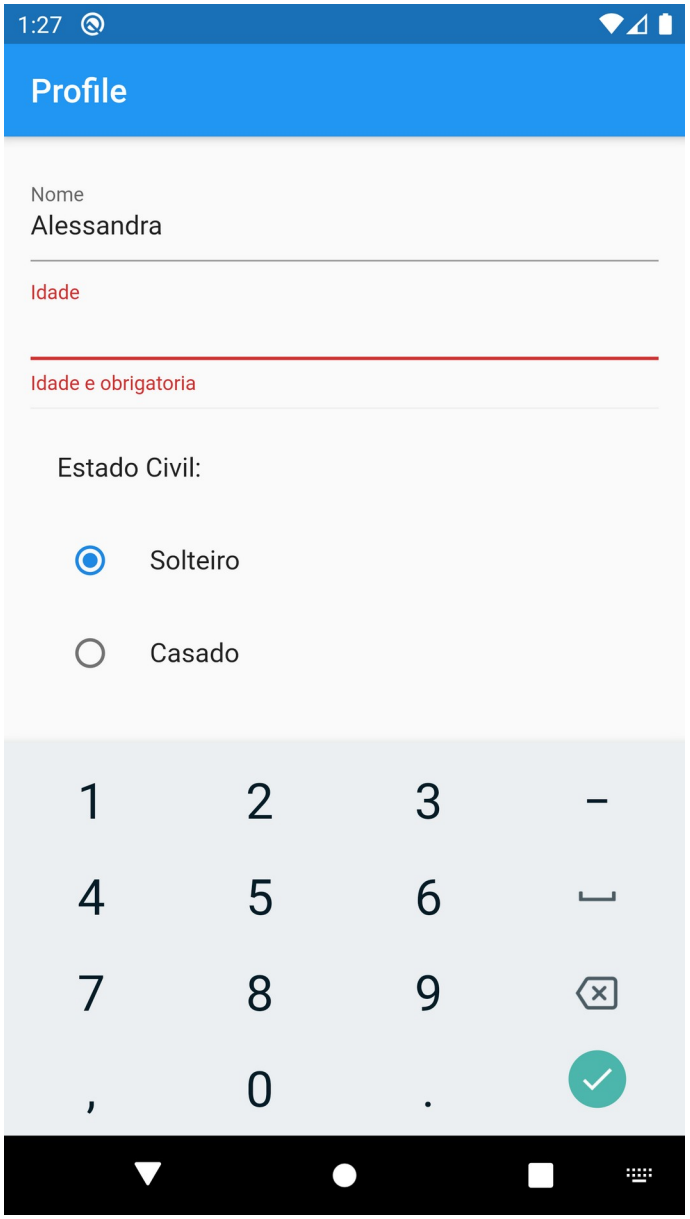
Focus

Altere o método para verificar e salvar os dados do formulário da classe _HomeState:

```
_submit() {
  if(_formStateKey.currentState.validate()) {
    _formStateKey.currentState.save();
    _user.save();
    _user.show();
    _scaffoldKey.currentState.showSnackBar(
        SnackBar(content: Text('Submitting form'))
    );
  }
  else {
    if ( !_nomeKey.currentState.validate() )
      _focusScopeNode.requestFocus(_nomeFocusNode);
    else if ( !_idadeKey.currentState.validate() )
      _focusScopeNode.requestFocus(_idadeFocusNode);
  }
}
```

Focus


1:27
Profile
Nome
Enter com o nome
Idade
Idade e obrigatoria
Estado Civil:
Solteiro
GIF
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
,
.


1:27
Profile
Nome
Alessandra
Idade
Idade e obrigatoria
Estado Civil:
Solteiro
Casado
1 2 3 -
4 5 6
7 8 9
, 0 .

Stepper

Stepper permite a divisão de um formulário em várias partes, fazendo o gerenciamento da exibição dessas partes como passos do formulário.

Crie uma nova aplicação Flutter com o nome **form_setpper**. Crie os mesmos diretórios criados para a aplicação anterior e copie os arquivos dentro desses diretórios.

Acrescente as chaves para os widgets do estado civil e atuação dentro da classe **_HomeState**:

```
  // Chave para o widget da idade
  final GlobalKey<FormFieldState> _idadeKey =
GlobalKey<FormFieldState>();
  // Chave para o widget do estado civil
  final GlobalKey<FormFieldState> _estcivKey =
GlobalKey<FormFieldState>();
  // Chave para o widget da atuacao
  final GlobalKey<FormFieldState> _atuacaoKey =
GlobalKey<FormFieldState>();
```

Stepper

Acrescente as variáveis para gerenciar a orientação e a página corrente do Stepper dentro da classe _HomeState:

```
// Gerenciar o foco do widget para nome
FocusNode _nomeFocusNode;
// Gerenciar o foco do widget para idade
FocusNode _idadeFocusNode;
// Orientacao do stepper
StepperType _stepperType = StepperType.vertical;
// Step corrente
int _current_step = 0;
```

Stepper

Substitua o child do Form por:

```
child: Stepper(
  currentStep: this._current_step,
  type: _stepperType,
  steps: <Step>[
  ],
  onStepTapped: (step) =>
    setState(() => _current_step = step),
  onStepContinue: () {
    setState(() {
      if (_current_step < 2) {
        _current_step = _current_step + 1;
      } else {
        _current_step = 0;
      }
    });
  },
```

Stepper

```
  onStepCancel: () {
    setState(() {
      if (_current_step > 0) {
        _current_step = _current_step - 1;
      } else {
        _current_step = 0;
      }
    });
  },
),
```

Stepper

Acrescente o passo de identificação dentro do steps do Stepper:

```
  Step(
    title: Text('Identificacao'),
    content: Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[
        TextFormField(
          key: _nomeKey,
          focusNode: _nomeFocusNode,
          initialValue: _user.nome,
          decoration: InputDecoration(labelText: 'Nome'),
          validator: (value) => _validateNome(value),
          onSaved: (value) => setState(() => _user.nome =
value),
        ),
```

Stepper

```
          TextFormField(
            key: _idadeKey,
            focusNode: _idadeFocusNode,
            initialValue:
              (_user.idade ?? 0)>=18 ?
_user.idade.toString() : '',
            inputFormatters:
              [WhitelistingTextInputFormatter.digitsOnly],
            keyboardType: TextInputType.number,
            decoration: InputDecoration(labelText: 'Idade'),
            validator: (value) => _validateIdade(value),
            onSaved: (value) =>
              setState(() => _user.idade =
int.tryParse(value)),
          ),
        ]
      ),
      isActive: true,
    ),
```

Stepper

Acrescente o passo de dados complementares dentro do steps do Stepper:

```
Step(
  title: Text('Dados'),
  content: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      FormField(
        key: _estcivKey,
        validator: (value) => _validateEstCiv(value),
        onSaved: (value) =>
          setState(() => _user.estciv = value),
        initialValue: _user.estciv,
        builder: (FormFieldState<String> state) {
          return Column(children: <Widget>[
            ListTile(
              title: Text('Estado Civil:'),
            ),
```

Stepper

```
                RadioListTile<String>(
                  title: const Text('Solteiro'),
                  value: 'solteiro',
                  groupValue: state.value,
                   onChanged: (value) =>
state.didChange(value),
                ),
                RadioListTile<String>(
                  title: const Text('Casado'),
                  value: 'casado',
                  groupValue: state.value,
                  onChanged: (value) =>
state.didChange(value),
                ),
                RadioListTile<String>(
                  title: const Text('Outro'),
                  value: 'outro',
                  groupValue: state.value,
                  onChanged: (value) =>
state.didChange(value),
                ),
```

Stepper

```
        state.hasError ?
          Text(
            state.errorText,
            style: TextStyle(color: Colors.red),
          )
        :
          Container()
      ]);
  }
),
Divider(),
```

Stepper

```
FormField(
  key: _atuacaoKey,
  validator: (value) => _validateAtuacao(value),
  onSaved: (value) =>
    setState(() => _user.atuacao = value),
  initialValue: _user.atuacao,
  builder: (FormFieldState<String> state) {
    return Column(children: <Widget>[
      ListTile(
        title: Text('Atuacao'),
        trailing: DropdownButton<String>(
          value: state.value,
          icon: Icon(Icons.arrow_downward),
          iconSize: 24,
          elevation: 16,
          onChanged: (value) =>
state.didChange(value),
```

Stepper

```
                items: _user.Atuacao
                  .map<DropdownMenuItem<String>>((String
value) {
                    return DropdownMenuItem<String>(
                      value: value,
                      child: Text(value),
                    );
                  }
                ).toList(),
              ),
            ),
```

Stepper

```
            state.hasError ?
              Text(
                state.errorText,
                style: TextStyle(color: Colors.red),
              )
            :
              Container()
          ]);
      }
    ),
  ]
 ),
 isActive: true,
),
```

Stepper

Acrescente o passo de preferências dentro do steps do Stepper:

```
  Step(
    title: Text('Preferencias'),
    content: Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: <Widget>[
        FormField(
          onSaved: (value) =>
            setState(() => _user.newsletter = value),
          initialValue: _user.newsletter?? false,
          builder: (FormFieldState<bool> state) {
            return SwitchListTile(
              title: const Text('Enviar Newsletter'),
              value: state.value,
              onChanged: (bool value) =>
state.didChange(value)
            );
          }
        ),
        Divider(),
```

Stepper

```
        FormField(
          onSaved: (value) =>
            setState(() =>
              _user.preferencias[User.PreferenciasHardware]
= value
          ),
          initialValue: hardware?? false,
          builder: (FormFieldState<bool> state) {
            return CheckboxListTile(
              title: const Text('Hardware'),
              value: state.value,
              onChanged: (bool value) =>
state.didChange(value)
            );
          }
        ),
```

Stepper

```
        FormField(
          onSaved: (value) =>
            setState(() =>
        _user.preferencias[User.PreferenciasDesenvolvimento]
= value
          ),
          initialValue: desenvolvimento?? false,
          builder: (FormFieldState<bool> state) {
            return CheckboxListTile(
              title: const Text('Desenvolvimento'),
              value: state.value,
              onChanged: (bool value) =>
state.didChange(value)
            );
          }
        ),
```

Stepper

```
        FormField(
          onSaved: (value) =>
            setState(() =>
              _user.preferencias[User.PreferenciasRedes] =
value
          ),
          initialValue: redes?? false,
          builder: (FormFieldState<bool> state) {
            return CheckboxListTile(
              title: const Text('Redes'),
              value: state.value,
              onChanged: (bool value) =>
state.didChange(value)
            );
          }
        ),
      ]
    ),
    isActive: true,
  ),
```

Stepper

Acrescente o botão para mudar a orientação do Stepper no bottomNavigationBar do Scaffold:

```
bottomNavigationBar: ButtonBar(
  alignment: MainAxisAlignment.spaceEvenly,
  children: <Widget>[
    RaisedButton(
        onPressed: _switchStepType,
        child: Text('Orientacao')
    ),
    RaisedButton(
        onPressed: () => _user.show(),
        child: Text('Exibir')
    ),
```

Stepper

Acrescente o método para para mudar a orientação do Stepper na classe _HomeState:

```
//
// Altera a orientacao do stepper
//
_switchStepType() {
  setState(() =>
  _stepperType == StepperType.horizontal ?
    _stepperType = StepperType.vertical
  :
    _stepperType = StepperType.horizontal);
}
```

Stepper

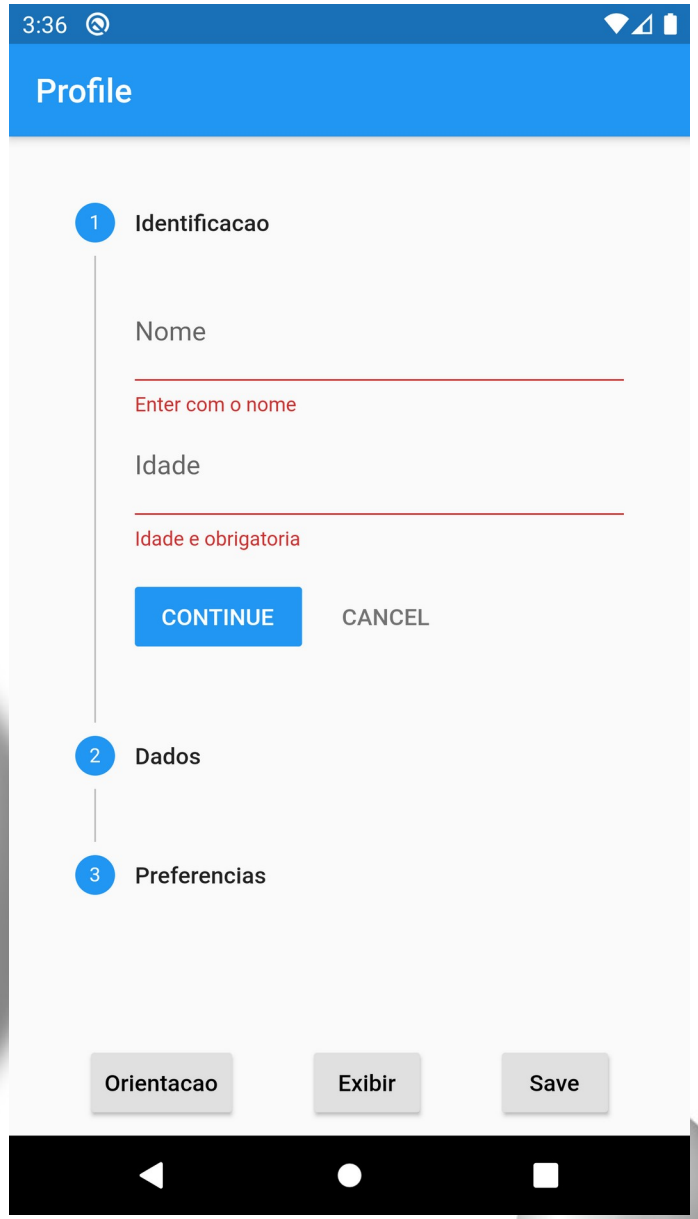
Altere o método para verificar e salvar os dados do formulário da classe _HomeState:

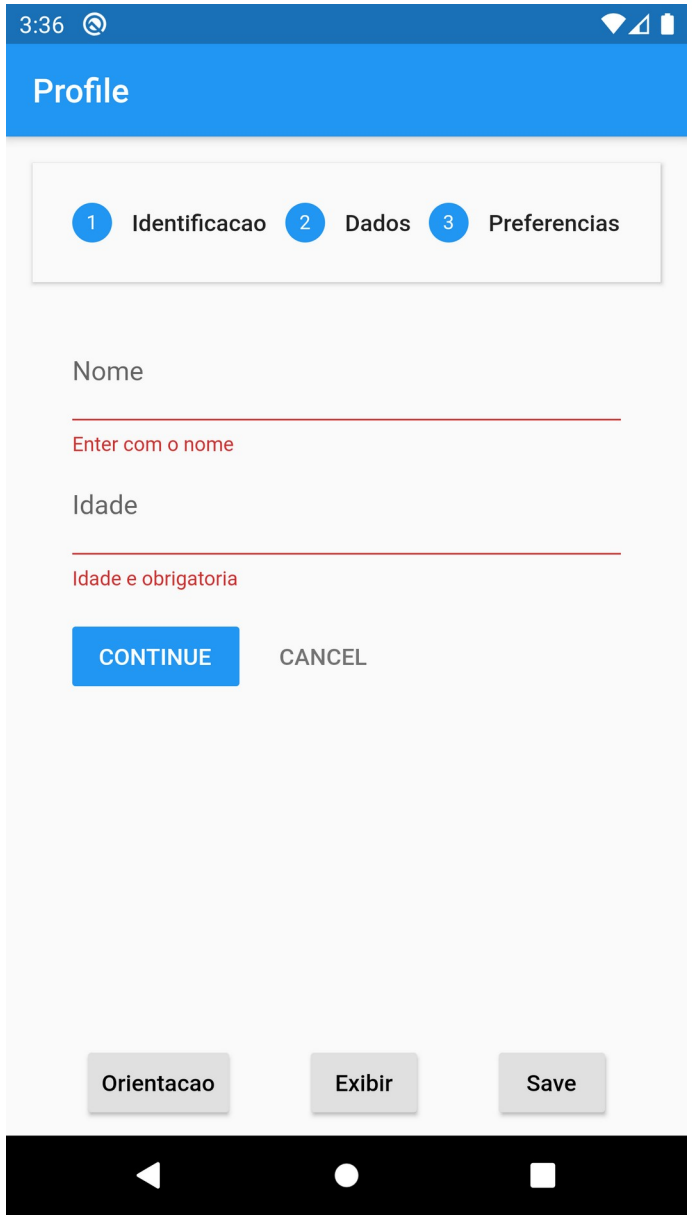
```
_submit() {
  if(_formStateKey.currentState.validate()) {
    _formStateKey.currentState.save();
    _user.save();
    _user.show();
    _scaffoldKey.currentState.showSnackBar(
        SnackBar(content: Text('Submitting form'))
    );
  }
```

Stepper

```
    else {
      if ( !_nomeKey.currentState.validate() ) {
        setState(() => _current_step = 0);
        _focusScopeNode.requestFocus(_nomeFocusNode);
      }
      else if ( !_idadeKey.currentState.validate() ) {
        setState(() => _current_step = 0);
        _focusScopeNode.requestFocus(_idadeFocusNode);
      }
      else if ( !_estcivKey.currentState.validate() ) {
        setState(() => _current_step = 1);
      }
      else if ( !_atuacaoKey.currentState.validate() ) {
        setState(() => _current_step = 1);
      }
    }
}
```

Stepper


3:36
Profile
1
Identificacao
Nome
Enter com o nome
Idade
Idade e obrigatoria
CONTINUE
CANCEL
2
Dados
3
Preferencias
Orientacao
Exibir
Save


3:36
Profile
1
Identificacao
2
Dados
3
Preferencias
Nome
Enter com o nome
Idade
Idade e obrigatoria
CONTINUE
CANCEL
Orientacao
Exibir
Save

Stepper

A divisão de um **Form** por um **Stepper** horizontal acarreta problemas com a validação e manutenção dos valores dos widgets baseados no estado do **Form**.

Uma alternativa é usar apenas **FormField** sem o **Form** e manter os valores dos widgets no modelo de dados do formulário.

Crie uma nova aplicação Flutter com o nome **form_setpper_horizontal**. Crie os mesmos diretórios criados para a aplicação anterior e copie os arquivos dentro desses diretórios.

Remova a chave para o **Form** da classe **_HomeState**:

```
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
```

Stepper

Altere a orientação inicial do Stepper:

```
// Orientacao do stepper
StepperType _stepperType = StepperType.horizontal;
```

Retire o Container, Builder e Form do body do Scaffold:

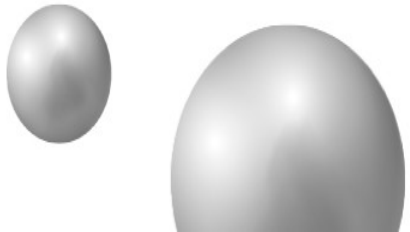
```
appBar: AppBar(title: Text('Profile')),
body: Stepper(
  currentStep: this._current_step,
```

Inclua no TextFormField do nome:

```
autovalidate: true,
onChanged: (value) => setState(() => _user.nome = value),
```

Retire do TextFormField do nome:

```
onSaved: (value) => setState(() => _user.nome = value),
```



Stepper

Inclua no TextFormField da idade:

```
autovalidate: true,
onChanged: (value) =>
  setState(() => _user.idade = int.tryParse(value)),
```

Retire do TextFormField da idade:

```
onSaved: (value) =>
  setState(() => _user.idade = int.tryParse(value)),
```

Inclua no FormField do estado civil:

```
autovalidate: true,
```

Retire do FormField do estado civil:

```
onSaved: (value) =>
  setState(() => _user.estciv = value),
```

Altere em todos RadioListTile do estado civil:

```
onChanged: (value) {
  _user.estciv = value;
  state.didChange(value);
},
```

Stepper

Inclua no FormField da atuação:

```
autovalidate: true,
```

Retire do FormField da atuação:

```
onSaved: (value) =>
  setState(() => _user.atuacao = value),
```

Altere no DropdownButtom da atuação:

```
onChanged: (value) {
  _user.atuacao = value;
  state.didChange(value);
},
```

Retire do FormField do newsletter:

```
onSaved: (value) =>
  setState(() => _user.newsletter = value),
```

Altere no SwitchListTile do newsletter:

```
onChanged: (value) {
  _user.newsletter = value;
  state.didChange(value); },
```

Stepper

Retire do FormField da preferência por hardware:

```
  onSaved: (value) =>
    setState(() =>
      _user.preferencias[User.PreferenciasHardware] = value
  ),
```

Altere no CheckboxListTile da preferência por hardware:

```
  onChanged: (value) {
    _user.preferencias[User.PreferenciasHardware] = value =
value;
    state.didChange(value);
  },
```

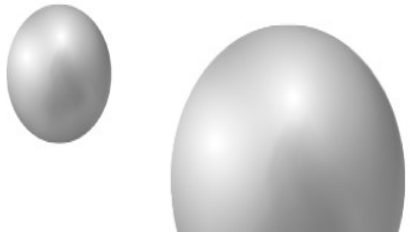
Stepper

Retire do FormField da preferência por desenvolvimento:

```
  onSaved: (value) =>
    SetState(() =>
      _user.preferencias[User.PreferenciasDesenvolvimento] =
value
  ),
```

Altere no CheckboxListTile da preferência por desenvolvimento:

```
  onChanged: (value) {
    _user.preferencias[User.PreferenciasDesenvolvimento] =
value = value;
    state.didChange(value);
  },
```



Stepper

Retire do FormField da preferência por redes:

```
  onSaved: (value) =>
    setState(() =>
      _user.preferencias[User.PreferenciasRedes] = value
  ),
```

Altere no CheckboxListTile da preferência por redes:

```
  onChanged: (value) {
    _user.preferencias[User.PreferenciasRedes] = value =
value;
    state.didChange(value);
  }
```

Stepper

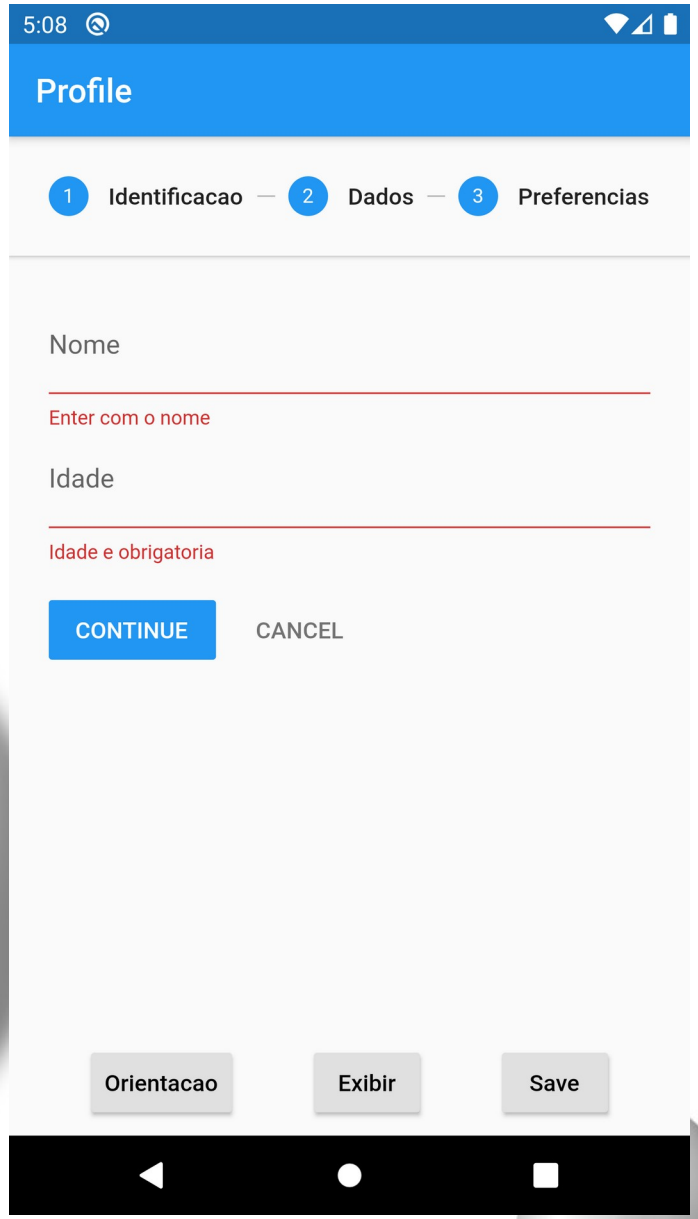
Altere o método para verificar e salvar os dados do formulário da classe _HomeState:

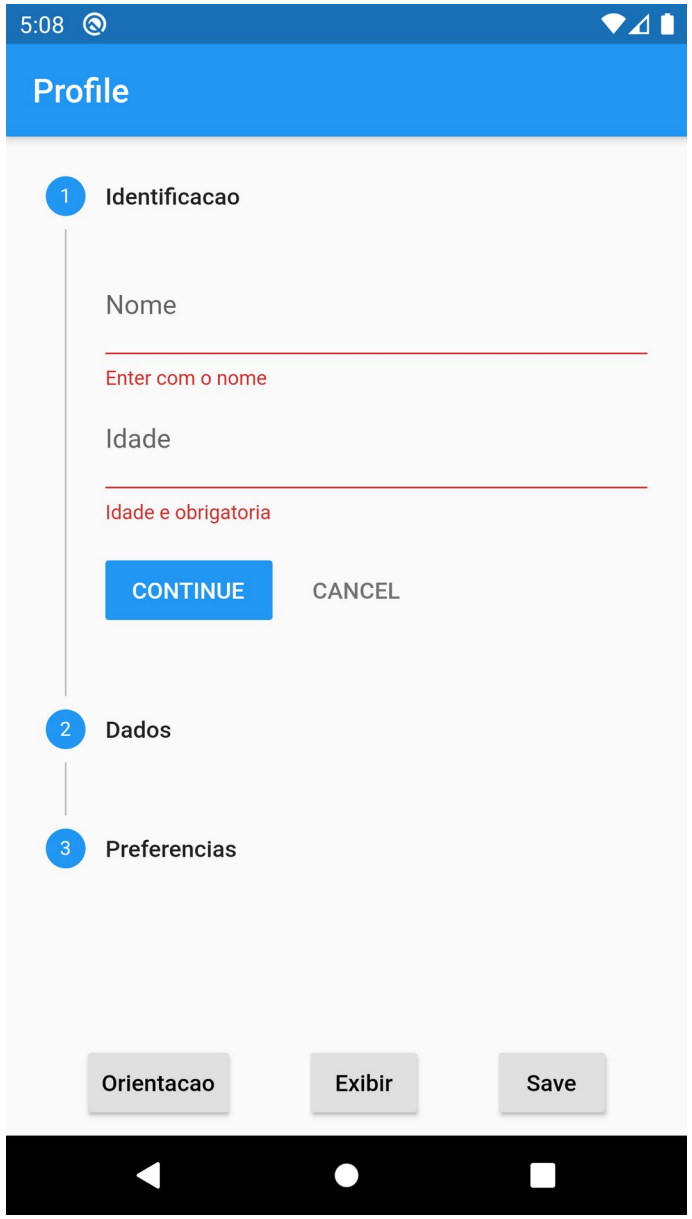
```
_submit() {
  if ( _validateNome(_user.nome)==null ) {
    if ( _validateIdade(_user.idade.toString())==null ) {
      if ( _validateEstCiv(_user.estciv)==null ) {
        if ( _validateAtuacao(_user.atuacao)==null ) {
          _user.show();
          _scaffoldKey.currentState.showSnackBar(
              SnackBar(content: Text('Submitting form')));
        } else {
          setState(() => _current_step = 1);
        }
      } else {
        setState(() => _current_step = 1);
      }
    } else {
      setState(() => _current_step = 0);
      _focusScopeNode.requestFocus(_idadeFocusNode);
    }
```

Stepper

```
    } else {
      setState(() => _current_step = 0);
      _focusScopeNode.requestFocus(_nomeFocusNode);
    }
  }
```

Stepper


5:08
Profile
1 Identificacao — 2 Dados — 3 Preferencias
Nome
Enter com o nome
Idade
Idade e obrigatoria
CONTINUE
CANCEL
Orientacao
Exibir
Save


5:08
Profile
1 Identificacao
Nome
Enter com o nome
Idade
Idade e obrigatoria
CONTINUE
CANCEL
2 Dados
3 Preferencias
Orientacao
Exibir
Save

Controller

TextEditingController permite o acesso ao conteúdo de um **TextField**. É possível recuperar ou alterar o conteúdo atual do **TextField**.

Crie uma nova aplicação Flutter com o nome **form_controller**.

Controller

Alterar o conteúdo de main.dart para:

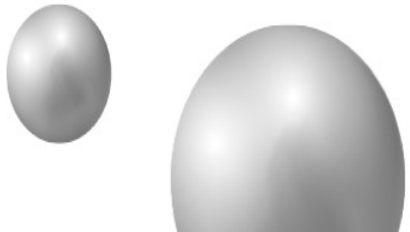
```
import 'package:flutter/material.dart';
import 'pages/home.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Retrieve Text Input',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity:
VisualDensity.adaptivePlatformDensity,
      ),
      home: Home(),
    );
  }
}
```

Controller

Crie o subdiretório **lib/pages** e o arquivo **home.dart** nesse subdiretório.

Acrescente a classe com a tela da aplicação:

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

class Home extends StatefulWidget {
  @override
  _HomeState createState() => _HomeState();
}
```



Controller

Acrescente a classe para gerenciar o estado do formulário:

```
class _HomeState extends State<Home> {

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Controller'),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: <Widget>[
          ],
        ),
      ),
    );
  }
}
```

Controller

Acrescente os controladores para o TextField 1 e TextField 2 dentro da classe _HomeState:

```
class _HomeState extends State<Home> {
  // Controlador para o TextField 1
  final text1Controller = TextEditingController();
  // Controlador para o TextField 2
  final text2Controller = TextEditingController();
```

Acrescente o método para inicializar o estado da classe _HomeState:

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

    text2Controller.addListener(_printLatestValue);
  }
```

Controller

Acrescente o método para descartar os objetos da classe _HomeState:

```
  @override
  void dispose() {
    text1Controller.dispose();
    text2Controller.dispose();

    super.dispose();
  }
```

Acrescente os TextFields no body do Scaffold:

```
            children: <Widget>[
              TextField(
                controller: text1Controller,
                onChanged: (text) => print("First text field:
$text"),
              ),
              TextField(
                controller: text2Controller,
              ),
```

Controller

Acrescente o botão para copiar o valor do TextField 1 após o body do Scaffold:

```
floatingActionButton: FloatingActionButton(
  onPressed: () => _copy(),
  tooltip: 'Copy!',
  child: Icon(Icons.content_copy),
),
```

Controller

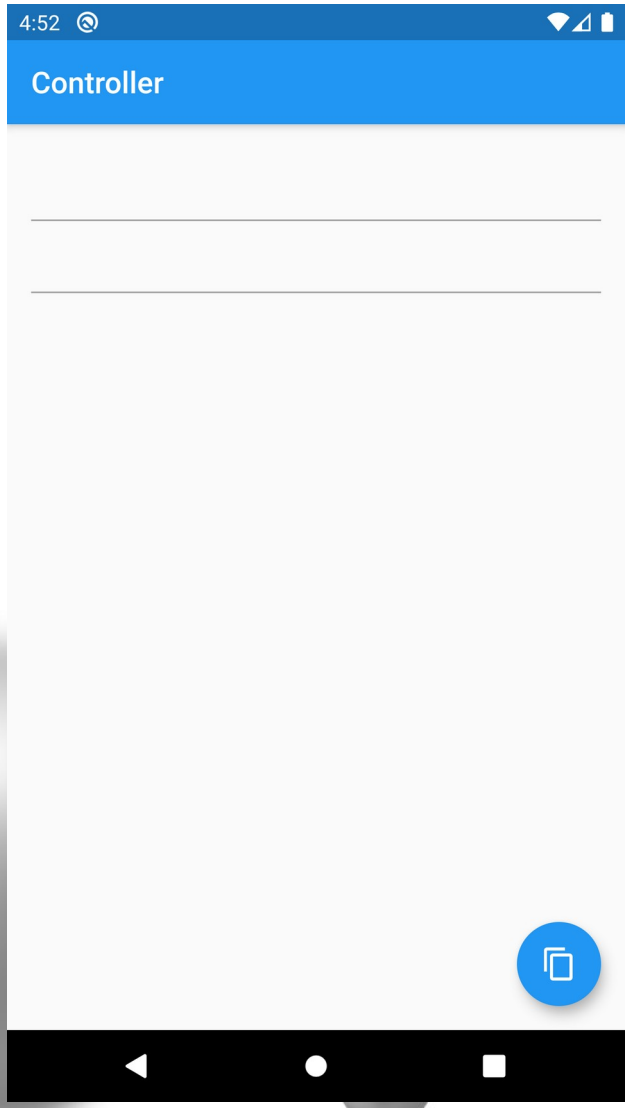
Acrescente o método para exibir o conteúdo do TextField 2 dentro da classe _HomeState:

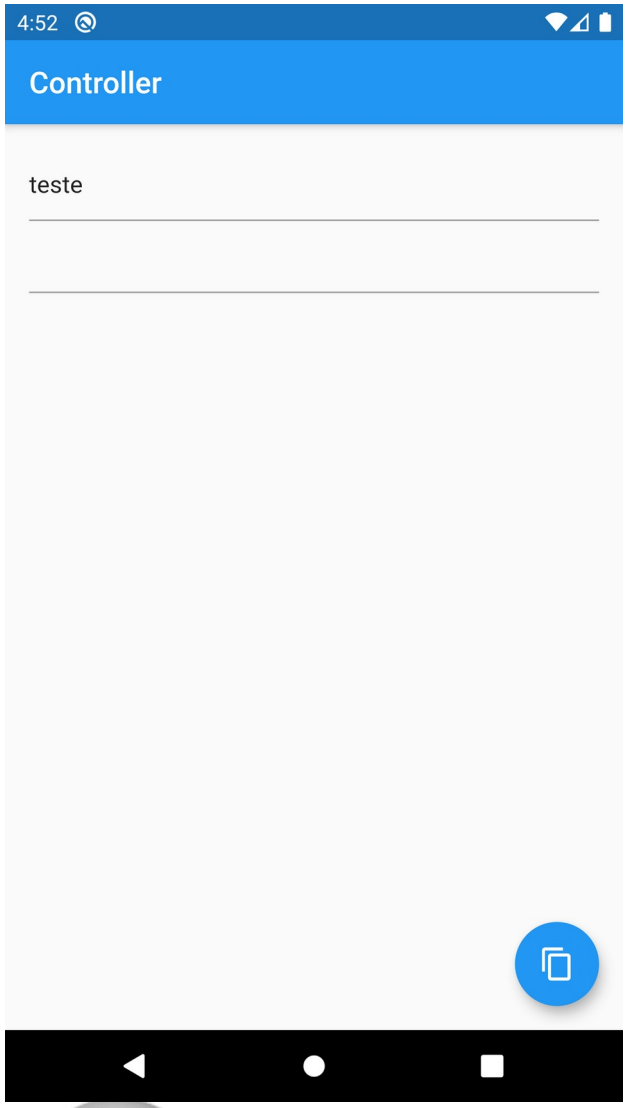
```
//
// Exibie o valor atual do TextField
//
_printLatestValue() {
  print("Second text field: ${text2Controller.text}");
}
```

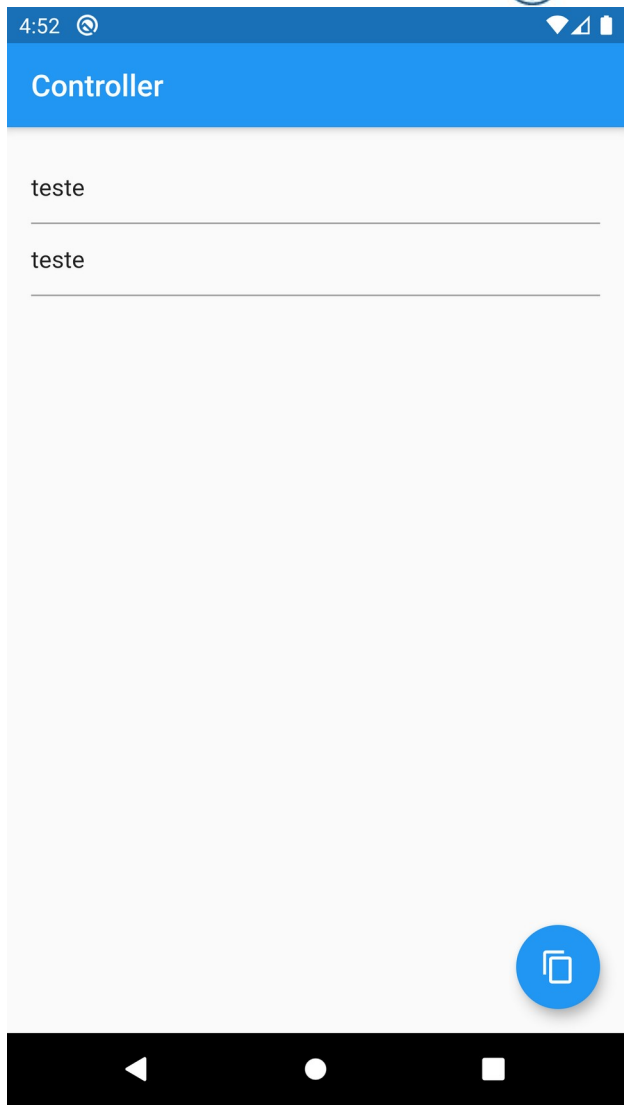
Acrescente o método para copiar o conteúdo do TextField 1 para o TextField 2 dentro da classe _HomeState:

```
//
// Copiar o TextField 1 para o TextField 2
//
_copy() {
  text2Controller.text = text1Controller.text;
}
```

Controller


4:52
Controller


4:52
Controller
teste


4:52
Controller
teste
teste