Методические указания к Практической работе №5
по дисциплине «Разработка кроссплатформенных мобильных приложений»
Тема работы: Формы ввода. Валидация данных

План практической работы:

- Знакомство с виджетами Form, TextField, TextFormField
- Знакомство с валидацией форм, использованием регулярных выражений для проверки значений полей
- Знакомство с управлением фокусами полей
- Выполнение практической работы №5 в соответствии с заданием

Последовательность выполнения практической работы:

1. Знакомство с виджетами Form, TextField, TextFormField

При взаимодействии с пользователями основными элементами остаются формы ввода, содержащие текстовые поля, списки, переключатели (радиокнопки) и флажки (чекбоксы).

После ввода перед использованием внесенных данных важно проверить:

- внесены ли данные в обязательные поля;
- соответствует ли содержимое значению поля (например, в поле возраст должно быть целое число или email содержит знак @);
- выбрано значение в группе переключателей и т.д.

И чтобы объединить и обработать все поля ввода, Flutter SDK предоставляет такой виджет, как Form.

Можно сказать, что виджет Form – это контейнер для полей формы, который оборачивает все поля ввода и делает их дочерними. Преимущество виджета форм в том, что, обращаясь к состоянию формы FormState, он предоставляет нам расширенные возможности, такие как сохранение данных, их сброс и валидацию, устраняя при этом необходимость обработки состояния для каждого поля отдельно.

Виджет TextField

Базовый виджет ввода текста без валидации. Используется, если просто нужно получить текст (например, строка поиска, ввод комментария), возможно использование без виджета Form

Некоторые параметры конструктора:

- decoration: - задает стилизацию виджета с помощью объекта InputDecoration
- enabled: true/false - указывает, доступно ли для ввода текстовое поле (при значении false поле для ввода недоступно)
- expands: true/false - указывает, будет ли виджет заполнять все доступное пространство контейнера
- focusNode - управляет фокусом (например, чтобы открыть клавиатуру или переключиться на следующее поле)
- inputFormatters – задает ограничители ввода (например, только цифры, маски)
- keyboardType: устанавливает тип клавиатуры в виде объекта TextInputType (например, TextInputType.number или TextInputType.emailAddress)
- maxLength: максимальное количество символов, которое можно ввести в текстовое поле
- maxLines: максимальное количество строк
- minLines: минимальное количество строк
- obscureText: true/false - указывает, будет ли виджет скрывать с помощью маски вводимые символы (например, при вводе пароля)
- obscuringCharacter: задает символы, которые применяются в качестве маски для вводимых символов, если параметр obscureText равен true.
- readOnly: true/false - указывает, можно ли изменять текст в виджете
- style: стиль вводимого текста в виде объекта TextStyle
- textAlign: задает горизонтальное выравнивание текста в виде объекта TextAlign
- textAlignVertical: задает вертикальное выравнивание текста в виде объекта TextAlignVertical
- textDirection: устанавливает направление текста с помощью объекта TextDirection
- textInputAction – отвечает за кнопки на клавиатуре (done, next, search)

Оформление поля ввода

С помощью параметра **decoration**, который представляет объект **InputDecoration**, можно стилизовать виджет TextField.

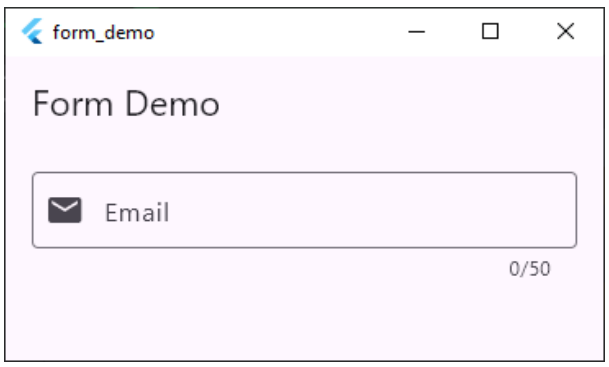
Некоторые основные параметры конструктора класса InputDecoration:

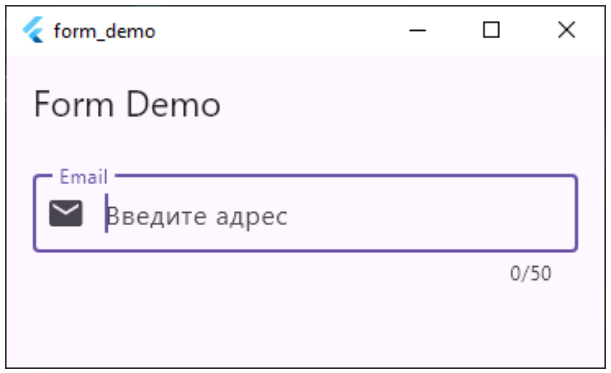
- border: устанавливает границу виджета в виде объекта InputBorder. По умолчанию виджет имеет нижнюю границу. Если надо полностью убрать границу, то можно использовать значение InputBorder.none

  Для создания границы можно применять еще два класса: OutlineInputBorder (создает границу вокруг элемента) и UnderlineInputBorder (создает только нижнюю границу).
- contentPadding: устанавливает отступы от границ виджета при вводе текста, представляет объект EdgeInsetsGeometry
- fillColor: устанавливает объект Color для заполнения виджета цветом
- filled: true/false - указывает, будет ли применяться заполнение виджета цветом из параметра
- fillColor (заполнение цветом применяется при значении true)
- hintText: приглашение для ввода текста, которое появляется в виджете при отсутствии в нем текста
- hintStyle: стиль текста из параметра hintText в виде объекта TextStyle
- icon: виджет (объект Widget), который представляет иконку, отображаемую перед виджетом
- prefixIcon — иконка внутри поля слева, перед текстом.
- suffixIcon — иконка внутри поля справа, после текста.
- prefix — виджет (иконка или текст), отображается перед вводимым текстом, прямо в строке.
- suffix — виджет (иконка или текст), отображается после вводимого текста, прямо в строке.
- prefixIconConstraints — ограничения размеров для prefixIcon.
- suffixIconConstraints — ограничения размеров для suffixIcon.
- helperText: вспомогательный текст, который появляется снизу справа от виджета и который указывает, как вводимое значение будет использоваться
- helperStyle: стиль текста из параметра helperText в виде объекта TextStyle
- hintText – задает текст подсказки в поле ввода
- labelText: текст метки, которая появляется сверху виджета и описывает предназначение текстового поля
- labelStyle: стиль текста из параметра labelText в виде объекта TextStyle

Пример

```
class _MyFormState extends State {
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(16),
      child: TextField(
        decoration: const InputDecoration(
          labelText: "Email",
          hintText: "Введите адрес",
          border: OutlineInputBorder(),
          prefixIcon: Icon(Icons.email),
        ), InputDecoration
        keyboardType: TextInputType.emailAddress,
        textInputAction: TextInputAction.next,
        onSubmitted: (text) {
          print("Текущий ввод: $text");
        },
        obscureText: false,
        maxLength: 50,
      ), TextField
    ); Padding
  }
}
```





Виджет TextFormField

Виджет TextFormField оборачивает виджет TextField и интегрирует его с окружающей формой. В результате чего мы получаем дополнительные функции, такие как: проверка поля на валидность и интеграцию с другими виджетами FormField

Дополнительные параметры TextFormField (которых нет в TextField)

- validator – принимает функцию String? Function(String?)?

Проверяет введённое значение и возвращает текст ошибки (или null, если всё корректно) Используется при вызове FormState.validate()

- onSaved – принимает функцию void Function(String?)?

Вызывается при сохранении формы (FormState.save()), позволяет обработать и сохранить введённое значение.

- autovalidateMode – управляет автоматической валидацией:

✓ AutovalidateMode.disabled — валидация только вручную (по умолчанию)

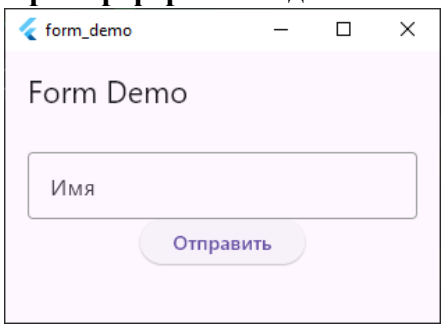
✓ AutovalidateMode.onUserInteraction — валидация запускается при вводе пользователем

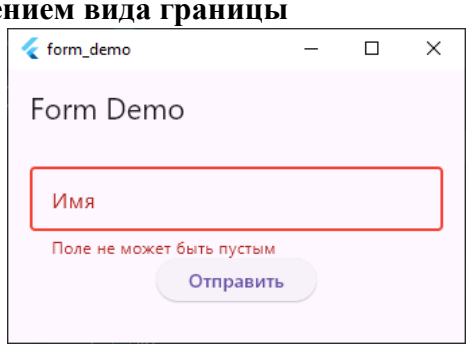
✓ AutovalidateMode.always — валидация срабатывает сразу

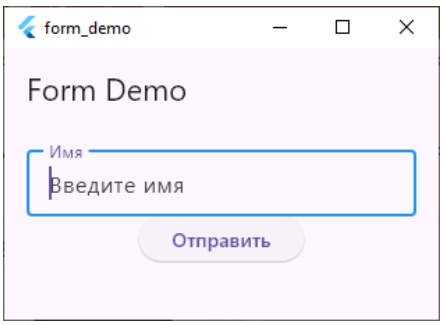
Виды границ (border) для разных состояний поля:

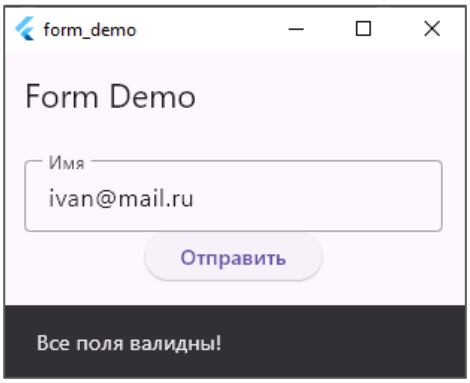
- enabledBorder – граница, когда поле активно, но не в фокусе
- focusedBorder – граница фокуса, когда поле в фокусе (курсор внутри)
- errorBorder – граница ошибки, когда поле не прошло валидацию
- focusedErrorBorder – граница, когда поле в фокусе и есть ошибка
- disabledBorder – граница, когда поле отключено (enabled: false)
- border – базовая граница по умолчанию (если не указаны другие).

Пример формы ввода с изменением вида границы









Программный код:

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

void main() => runApp(
    MaterialApp(
        debugShowCheckedModeBanner: false,
        home: Scaffold(
            appBar: AppBar(title: Text('Form Demo')),
            body: MyForm()
        )
    )
);
```

```dart
class MyForm extends StatefulWidget {
  @override
  _MyFormState createState() => _MyFormState();
}

class _MyFormState extends State {
  final _formKey = GlobalKey<FormState>();
  @override
  Widget build(BuildContext context) {
    return Padding(
      padding: EdgeInsets.all(16),
      child: Form(
        key: _formKey,
        child: Column(
          children: [
            TextFormField(
              decoration: InputDecoration(
                labelText: "Имя",
                hintText: "Введите имя",
                border: OutlineInputBorder(),

                enabledBorder: OutlineInputBorder(
                  borderSide: BorderSide(color: Colors.grey, width: 1),
                ),

                focusedBorder: OutlineInputBorder(
                  borderSide: BorderSide(color: Colors.blue, width: 2),
                ),

                errorBorder: OutlineInputBorder(
                  borderSide: BorderSide(color: Colors.red, width: 2),
                ),
              ),
              validator: (value) {
                if (value == null || value.isEmpty) {
                  return "Поле не может быть пустым";
                }
                return null;
              },
            ),
            ElevatedButton(
              onPressed: () {
                if (_formKey.currentState!.validate()) {
                  // Все поля валидны
                  ScaffoldMessenger.of(context).showSnackBar(
                    SnackBar(content: Text("Все поля валидны!")),
                  );
                }
              },
              child: Text('Отправить'),
            ),
          ],
        ),
      )
    );
  }
}
```

2. **Знакомство с валидацией форм, использованием регулярных выражений для проверки значений полей**

Валидация форм во Flutter

В приложении Flutter существует множество способов проверки формы, таких как использование TextEditingController. Но работа с текстовым контроллером для каждого объекта ввода может быть затруднительной в больших приложениях. Виджет **Form** предоставляет нам удобный способ проверки вводимых пользователем данных.

В Form ввод проверяется в функции submit (функция, которая вызывается после того, как пользователь ввел все данные, но условие применяется в каждом поле TextFormField с помощью валидатора)

Регулярные выражения

В validator можно использовать RegExp (класс регулярных выражений в Dart) для проверки значения.

Регулярное выражение (regex) — это шаблон для поиска или проверки строки.

Работа с регулярными выражениями в validator:

1) Создание объекта: final reg = RegExp(r'шаблон');

2) Проверка ввода: reg.hasMatch(value) возвращает true/false.

3) В validator возвращаем текст ошибки, если false, или null, если всё корректно.

Пример

```
validator: (value) {
  if (value == null || value.isEmpty) {
     return "Введите пароль";
   }
   final reg = RegExp(r'^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d]{6,}$');
   if (!reg.hasMatch(value)) {
       return "Пароль должен быть не менее 6 символов и содержать буквы (хотя бы одну) и цифры (хотя бы одну)";
    }
      return null;
}
```

3. Знакомство с управлением фокусами полей

Управление фокусом полей формы

Управление фокусом является фундаментальным инструментом для создания форм с интуитивно понятным интерфейсом и позволяет программно управлять фокусом в полях формы.

Основные инструменты

1) FocusNode - объект, который «привязывается» к конкретному TextFormField и управляет фокусом (активно/не активно).

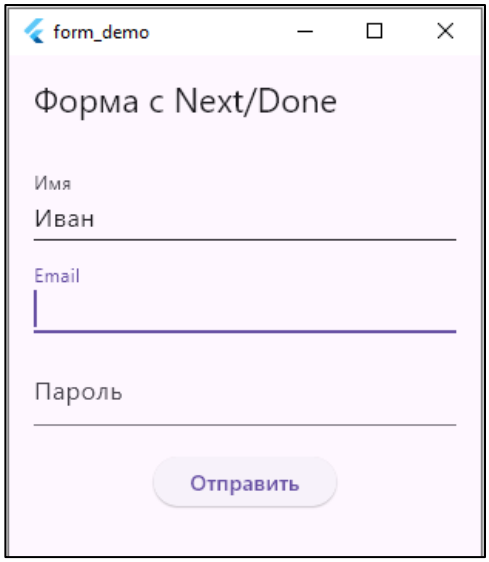
2) FocusScope.of(context) - менеджер фокусов, позволяет переключаться между полями: requestFocus(node) или unfocus().

3) TextInputAction меняет кнопку на клавиатуре (например, next, done), удобно для перехода к следующему полю.

2 способ

Использование FocusScope.of(context).nextFocus() для автоматического перехода фокуса к следующему доступному полю.

Пример

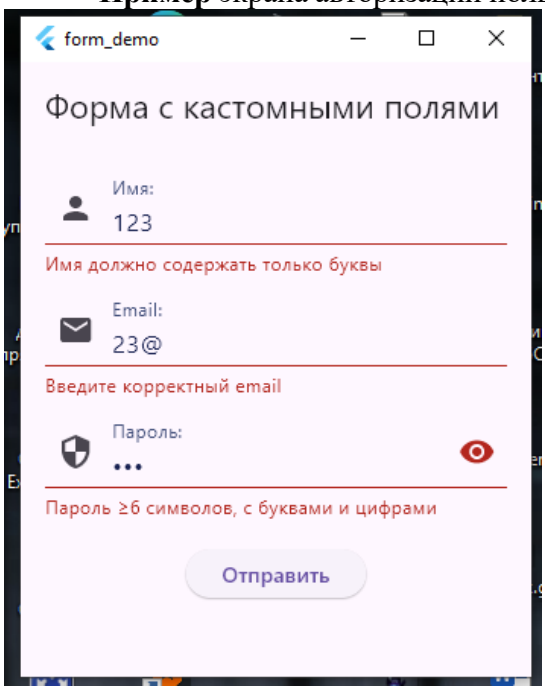


Программный код:

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

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

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      home: const MyFormPage(),
    );
  }
}

class MyFormPage extends StatefulWidget {
  const MyFormPage({super.key});

  @override
  State<MyFormPage> createState() => _MyFormPageState();
}

class _MyFormPageState extends State<MyFormPage> {
  final _formKey = GlobalKey<FormState>();

  void _submit() {
    if (_formKey.currentState!.validate()) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text("Форма успешно отправлена")),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Управление фокусами")),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              TextFormField(
                decoration: const InputDecoration(labelText: "Имя"),
                textInputAction: TextInputAction.next,
                onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
              ),

              const SizedBox(height: 12),

              TextFormField(
                decoration: const InputDecoration(labelText: "Email"),
                textInputAction: TextInputAction.next,
                keyboardType: TextInputType.emailAddress,
                onFieldSubmitted: (_) => FocusScope.of(context).nextFocus(),
              ),

              const SizedBox(height: 12),

              TextFormField(
                decoration: const InputDecoration(labelText: "Пароль"),
                obscureText: true,
                textInputAction: TextInputAction.done,
                onFieldSubmitted: (_) => _submit(),
              ),
```

```
              const SizedBox(height: 20),

              ElevatedButton(
                onPressed: _submit,
                child: const Text("Отправить"),
              )
            ],
          ),
        ),
      ),
    );
  }
}
```

Пример экрана авторизации пользователя



Программный код кастомного TextFormField (файл text_field.dart)

```
import 'package:flutter/material.dart';
enum InputFieldType {name, email, password}

class CustomTextFormField extends StatefulWidget {
  final InputFieldType inputType;
  final String labelText;
  final String hintText;
  final Icon prefIcon;
  final TextEditingController controller;

  const CustomTextFormField({
    super.key,
    required this.inputType,
    required this.labelText,
    required this.hintText,
    required this.prefIcon,
    required this.controller,
  });
  @override
  State<CustomTextFormField> createState() => _CustomTextFieldState();
 }

class _CustomTextFieldState extends State<CustomTextFormField> {
  bool _hidePassword = true;

  String? _validator(String? value) {
    if (value == null || value.isEmpty) {
```

```
      return "Поле не может быть пустым";
    }

    switch (widget.inputType) {
      case InputFieldType.name:
        final reg = RegExp(r'^[A-Za-zA-Яа-я\s]+$'); // только буквы и пробелы
        if (!reg.hasMatch(value)) {
          return "Имя должно содержать только буквы";
        }
        break;
      case InputFieldType.email:
        break;

      case InputFieldType.password:
        break;
    }
    return null;
  }
  @override
  Widget build(BuildContext context) {
    final isPassword = widget.inputType == InputFieldType.password;

    return TextFormField(
      controller: widget.controller,
      obscureText: isPassword && _hidePassword,
      keyboardType: widget.inputType == InputFieldType.email
          ? TextInputType.emailAddress
          : TextInputType.text,
      style: const TextStyle(
          color: Color(0xFF192252),
          fontWeight: FontWeight.w400
      ),
      decoration: InputDecoration(
        labelText: widget.labelText,
        labelStyle: TextStyle(
          color: const Color(0xFF424F7B),
          fontSize: 16,
        ),
        hintText: widget.hintText,
        prefixIcon: widget.prefIcon,
        suffixIcon: isPassword
            ? IconButton(
          icon: Icon(
            _hidePassword ? Icons.visibility : Icons.visibility_off,
          ),
          onPressed: () {
            setState(() => _hidePassword = !_hidePassword);
          },
        )
            : null,
        enabledBorder: OutlineInputBorder(
          borderSide: BorderSide(
            color: const Color(0xFF424F7B),
            width: 2.0,
          ),
          borderRadius: BorderRadius.circular(10.0),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(
            color: const Color(0xFF424F7B),
            width: 2.0,
          ),
          borderRadius: BorderRadius.circular(10.0),
        ),
      ),
      validator: _validator,
    );
  }
}
```

Программный код экрана авторизации (файл login_pade.dart)

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

class LoginPage extends StatelessWidget {
  LoginPage({super.key});

  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _emailController = TextEditingController();
  final _passwordController = TextEditingController();

  void _submit(BuildContext context) {
    if (_formKey.currentState!.validate()) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(
          "Имя: ${_nameController.text}, "
              "Email: ${_emailController.text}, "
              "Пароль: ${_passwordController.text}",
        )),
      );
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text("Форма с кастомными полями")),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Form(
          key: _formKey,
          child: Column(
            children: [
              CustomTextFormField(
                inputType: InputFieldType.name,
                labelText: "Имя:",
                hintText: "Введите ваше имя",
                prefIcon: const Icon(Icons.person),
                controller: _nameController,
              ),
              const SizedBox(height: 12),
              CustomTextFormField(
                inputType: InputFieldType.email,
                labelText:"Email:",
                hintText: "example@mail.com",
                prefIcon: const Icon(Icons.email),
                controller: _emailController,
              ),
              const SizedBox(height: 12),
              CustomTextFormField(
                inputType: InputFieldType.password,
                labelText: "Пароль:",
                hintText: "Пароль не менее 6 символов",
                prefIcon: const Icon(Icons.security),
                controller: _passwordController,
              ),
              const SizedBox(height: 20),
              ElevatedButton(
                onPressed: () => _submit(context),
                child: const Text("Отправить"),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

4. Задание на практическую работу

Изменить приложение, созданное на предыдущей практической работе, выполнив следующие действия:

1) Добавить в проект в папку lib папку pages, в которой создать файлы login_page.dart (экран авторизации), register_page.dart (экран регистрации), profile_page.dart(экран профиля)
2) Добавить в папку lib папку widgets, в которой создать файл text_field.dart с кастомным виджетом TextFormField, позволяющим отображать любое текстовое поле (полное имя, email, пароль, подтверждение пароля) с валидацией вводимых данных:
   - ✓ Поле не может быть пустым и неопределенным
   - ✓ Полное имя может содержать только буквы и пробелы
   - ✓ Email – корректный формат
   - ✓ Пароль не менее 6 символов, обязательно содержит буквы, цифры и символы (+, _, -)

   Виджет имеет соответствующую иконку слева, а также для паролей иконку отображения скрытого текста справа
3) Реализуйте переход фокуса к следующему доступному полю
4) Создайте экраны авторизации, регистрации и профиля пользователя согласно макетам ниже.

   P.S. Для запуска на выполнение соответствующего файла добавьте в него метод:

```dart
void main() {
  runApp(MaterialApp(
    debugShowCheckedModeBanner: false,
    home: LoginPage(),
  ));
}
```

   Выполнение файла (на примере файла login_page.dart): Правая кнопка мыши - Run 'login_page.dart'