Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ: «Информатика и системы управления»
КАФЕДРА: «Теоретическая информатика и компьютерные технологии»

Лабораторная работа №5
«Отправка и получение значений по MQTT»
по курсу
«Разработка мобильных приложений»

Выполнил:
студент группы ИУ9-72Б
Караник А.А.

Проверено:
Посевин Д.П.

Москва, 2024

Цель работы

Цель данной работы состоит в том, чтобы научиться реализовывать передачу и получение данных с помощью протокола MQTT в flutter мобильном приложении. В рамках лабораторной работы необходимо создать три текстовых поля для ввода данных и настроить два виджета: один для отправки значений по MQTT, а другой --- для получения данных. Также дополнить выпадающее меню в AppBar, которое будет содержать переключение между этими виджетами.

Реализация

Исходный код программы:

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:math';
import 'dart:convert';
import 'dart:async';
import 'dart:io';
import 'package:mqtt_client/mqtt_client.dart';
import 'package:mqtt_client/mqtt_server_client.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'lab5',
      home: HomeScreen(),
    );
  }
}

class HomeScreen extends StatefulWidget {
  @override
  _HomeScreenState createState() => _HomeScreenState();
}

class _HomeScreenState extends State<HomeScreen> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('lab5'),
        actions: [
          PopupMenuButton<String>(
            onSelected: (value) {
              if (value == 'lab2') {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => Lab2Screen()),
                );
              } else if (value == 'lab3') {
                Navigator.push(
                  context,
                  MaterialPageRoute(builder: (context) => Lab3Screen()),
                );
              } else if (value == 'anim') {
```

```
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => AnimScreen()));
              } else if (value == 'mqtt1') {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => Mqtt1Screen()));
              }  else if (value == 'mqtt2') {
                Navigator.push(context,
                    MaterialPageRoute(builder: (context) => Mqtt2Screen()));
              }
            },
            itemBuilder: (BuildContext context) {
              return [
                PopupMenuItem(
                  value: 'lab2',
                  child: Text('Lab 2'),
                ),
                PopupMenuItem(
                  value: 'lab3',
                  child: Text('Lab 3'),
                ),
                PopupMenuItem(
                  value: 'anim',
                  child: Text('Anim'),
                ),
                PopupMenuItem(
                  value: 'mqtt1',
                  child: Text('mqtt1'),
                ),
                PopupMenuItem(
                  value: 'mqtt2',
                  child: Text('mqtt2'),
                ),
              ];
            },
          ),
        ],
      ),
      body: Center(
        child: Text('Выберите лабораторную работу из меню'),
      ),
    );
  }
}

class Lab2Screen extends StatefulWidget {
  @override
  _Lab2ScreenState createState() => _Lab2ScreenState();
}

class _Lab2ScreenState extends State<Lab2Screen> {
  String response = "Здесь будет ответ";
  bool isSwitched = false;

  Future<void> requestOff() async {
    final response = await http
        .get(Uri.parse('http://iocontrol.ru/api/sendData/karanik/value/0'));

    if (response.statusCode == 200) {
      setState(() {
        this.response = jsonDecode(response.body).toString();
      });
    } else {
      setState(() {
        this.response = 'Failed';
      });
```

```
    }
  }

  Future<void> requestOn() async {
    final response = await http
        .get(Uri.parse('http://iocontrol.ru/api/sendData/karanik/value/1'));

    if (response.statusCode == 200) {
      setState(() {
        this.response = jsonDecode(response.body).toString();
      });
    } else {
      setState(() {
        this.response = 'Failed';
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("lab2"),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            // Переключатель on/off
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Text("OFF"),
                Switch(
                  value: isSwitched,
                  onChanged: (value) {
                    setState(() {
                      isSwitched = value;
                      if (isSwitched) {
                        requestOn();
                      } else {
                        requestOff();
                      }
                    });
                  },
                ),
                Text("ON"),
              ],
            ),
            SizedBox(height: 16),
            Text(response),
          ],
        ),
      ),
    );
  }
}

class Lab3Screen extends StatefulWidget {
  @override
  _Lab3ScreenState createState() => _Lab3ScreenState();
}

class _Lab3ScreenState extends State<Lab3Screen> {
```

```
int _counter = 0;
String _serverResponse = '';
final TextEditingController _textController = TextEditingController();

void _incrementCounter() {
  setState(() {
    _counter++;
  });
  _sendCounterToServer();
}

void _decrementCounter() {
  setState(() {
    _counter--;
  });
  _sendCounterToServer();
}

// Отправка значения на сервер (POST)
Future<void> _sendCounterToServer() async {
  final url = Uri.parse('http://194.67.88.154:8100/$_counter');
  try {
    final response = await http.post(url);

    if (response.statusCode == 200) {
      setState(() {
        _serverResponse = 'Значение отправлено: $_counter';
      });
    } else {
      setState(() {
        _serverResponse =
            'Не удалось отправить значение. Сервер ответил кодом статуса:
            ↪ ${response.statusCode}';
      });
    }
  } catch (e) {
    setState(() {
      _serverResponse = 'Ошибка отправки значения: $e';
    });
  }
}

Future<void> _sendValueToServer(int value) async {
  final url = Uri.parse('http://194.67.88.154:8100/$value');
  try {
    final response = await http.post(url);

    if (response.statusCode == 200) {
      setState(() {
        _serverResponse = 'Значение отправлено: $value';
        _counter = value;
      });
    } else {
      setState(() {
        _serverResponse =
            'Не удалось отправить значение. Сервер ответил кодом статуса:
            ↪ ${response.statusCode}';
      });
    }
  } catch (e) {
    setState(() {
      _serverResponse = 'Ошибка отправки значения: $e';
    });
  }
}
```

```dart
// Получение значения с сервера (GET)
Future<void> _getValueFromServer() async {
  final url = Uri.parse('http://194.67.88.154:8100');
  try {
    final response = await http.get(url);

    if (response.statusCode == 200) {
      setState(() {
        _counter = int.parse(response.body);
        _serverResponse = 'Счетчик обновлен с сервера: $_counter';
      });
    } else {
      setState(() {
        _serverResponse =
            'Не удалось получить значение. Сервер ответил кодом статуса:
            ↪ ${response.statusCode}';
      });
    }
  } catch (e) {
    setState(() {
      _serverResponse = 'Ошибка получения значения: $e';
    });
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text("lab3")),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          const SizedBox(height: 20),
          TextField(
            controller: _textController,
            decoration: const InputDecoration(
              labelText: 'Введите init value',
              border: OutlineInputBorder(),
            ),
          ),
          const SizedBox(height: 10),
          ElevatedButton(
            onPressed: () {
              _sendValueToServer(int.parse(_textController.text));
            },
            child: const Text('POST INIT'),
          ),
          const Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            style: Theme.of(context).textTheme.headlineMedium,
          ),
          const SizedBox(height: 20),
          ElevatedButton(
            onPressed: _getValueFromServer,
            child: const Text('GET Counter'),
          ),
          const SizedBox(height: 20),
          Text(
            _serverResponse,
            style: const TextStyle(color: Colors.lightGreen),
```

```
            ),
          ],
        ),
      ),
      floatingActionButton: Column(
        mainAxisAlignment: MainAxisAlignment.end,
        children: <Widget>[
          FloatingActionButton(
            onPressed: _incrementCounter,
            tooltip: 'Increment',
            child: const Icon(Icons.add),
          ),
          const SizedBox(height: 10),
          FloatingActionButton(
            onPressed: _decrementCounter,
            tooltip: 'Decrement',
            child: const Icon(Icons.remove),
          ),
        ],
      ),
    );
  }
}

class AnimScreen extends StatefulWidget {
  @override
  _AnimScreenState createState() => _AnimScreenState();
}

class _AnimScreenState extends State<AnimScreen>
    with SingleTickerProviderStateMixin {
  double a = 1;
  double b = 0;
  double c = 0;

  late AnimationController _controller;

  @override
  void initState() {
    super.initState();
    _controller =
        AnimationController(vsync: this, duration: Duration(seconds: 1))
          ..repeat();
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Anim'),
      ),
      body: Column(
        children: [
          Expanded(
            child: AnimatedBuilder(
              animation: _controller,
              builder: (context, child) {
                return CustomPaint(
                  painter: ParabolaPainter(a, b, c),
```

```
                child: Container(),
              );
            },
          ),
        ),
        _buildSlider('a', a, -10, 10, (val) => setState(() => a = val)),
        _buildSlider('b', b, -10, 10, (val) => setState(() => b = val)),
        _buildSlider('c', c, -10, 10, (val) => setState(() => c = val)),
      ],
    ),
  );
 }

 Widget _buildSlider(String label, double value, double min, double max,
     ValueChanged<double> onChanged) {
   return Padding(
     padding: const EdgeInsets.symmetric(horizontal: 20, vertical: 10),
     child: Row(
       children: [
         Text('$label: ', style: TextStyle(fontSize: 18)),
         Expanded(
           child: Slider(
             value: value,
             min: min,
             max: max,
             divisions: 100,
             label: value.toStringAsFixed(2),
             onChanged: onChanged,
           ),
         ),
       ],
     ),
   );
 }
}

class ParabolaPainter extends CustomPainter {
 final double a;
 final double b;
 final double c;

 ParabolaPainter(this.a, this.b, this.c);

 @override
 void paint(Canvas canvas, Size size) {
   final paint = Paint()
     ..color = Colors.blue
     ..strokeWidth = 2
     ..style = PaintingStyle.stroke;

   final centerX = size.width / 2;
   final centerY = size.height / 2;

   canvas.drawLine(Offset(0, centerY), Offset(size.width, centerY), paint);
   canvas.drawLine(Offset(centerX, 0), Offset(centerX, size.height), paint);

   paint.color = Colors.green;

   final path = Path();
   for (double x = -centerX; x <= centerX; x += 1) {
     double y = a * pow(x / 50, 2) + b * (x / 50) + c;
     if (x == -centerX) {
       path.moveTo(centerX + x, centerY - y * 50);
     } else {
       path.lineTo(centerX + x, centerY - y * 50);
```

```
      }
    }

    canvas.drawPath(path, paint);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) {
    return true;
  }
}

class Mqtt1Screen extends StatefulWidget {
  @override
  Mqtt1ScreenState createState() => Mqtt1ScreenState();
}

class Mqtt1ScreenState extends State<Mqtt1Screen> {
  final _formKey = GlobalKey<FormState>();
  String _valueA = "";
  String _valueB = "";
  String _valueC = "";

  final client = MqttServerClient('test.mosquitto.org', '');

  @override
  void initState() {
    super.initState();
    setupMqttClient();
  }

  Future<void> setupMqttClient() async {
    client.logging(on: true);
    client.setProtocolV311();
    client.keepAlivePeriod = 20;
    client.onDisconnected = onDisconnected;
    client.onConnected = onConnected;
    client.onSubscribed = onSubscribed;

    try {
      await client.connect();
    } catch (e) {
      print('Connection exception - $e');
      client.disconnect();
    }
  }

  void sendMessage(String topic, String message) {
    if (client.connectionStatus!.state == MqttConnectionState.connected) {
      final builder = MqttClientPayloadBuilder();
      builder.addString(message);
      client.publishMessage(topic, MqttQos.exactlyOnce, builder.payload!);
      print('Message "$message" sent to topic "$topic"');
    } else {
      print('MQTT Client is not connected');
    }
  }

  void onSubscribed(String topic) {
    print('Subscription confirmed for topic $topic');
  }

  void onDisconnected() {
    print('Disconnected from the broker');
  }
```

```
  void onConnected() {
    print('Connected to the broker');
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
        appBar: AppBar(
          title: Text('MQTT Publisher'),
        ),
        body: Container(
            padding: EdgeInsets.all(10.0),
            child: Form(
                key: _formKey,
                child: Column(
                  children: <Widget>[
                    TextFormField(
                     decoration: InputDecoration(labelText: 'Введите значение для топика
                       ↪ A'),
                      onSaved: (value) => _valueA = value!,
                    ),
                    TextFormField(
                     decoration: InputDecoration(labelText: 'Введите значение для топика
                       ↪ B'),
                      onSaved: (value) => _valueB = value!,
                    ),
                    TextFormField(
                     decoration: InputDecoration(labelText: 'Введите значение для топика
                       ↪ C'),
                      onSaved: (value) => _valueC = value!,
                    ),
                    SizedBox(height: 20.0),
                    ElevatedButton(
                      child: Text('Отправить'),
                      onPressed: () {
                        if (_formKey.currentState!.validate()) {
                          _formKey.currentState!.save();

                          sendMessage('IU9/test/a', _valueA);
                          sendMessage('IU9/test/b', _valueB);
                          sendMessage('IU9/test/c', _valueC);

                          ScaffoldMessenger.of(context).showSnackBar(SnackBar(
                            content: Text('Сообщения отправлены'),
                          ));
                        }
                      },
                      style: ElevatedButton.styleFrom(
                          padding: EdgeInsets.symmetric(
                              horizontal: 50, vertical: 20),
                          textStyle: TextStyle(
                              fontSize: 20, fontWeight: FontWeight.bold)),
                    ),
                  ],
                ))));
  }
}

class Mqtt2Screen extends StatefulWidget {
  @override
  Mqtt2ScreenState createState() => Mqtt2ScreenState();
}

class Mqtt2ScreenState extends State<Mqtt2Screen> {
```

```
// Переменные для хранения полученных значений
String _valueA = "Ожидание данных...";
String _valueB = "Ожидание данных...";
String _valueC = "Ожидание данных...";

final client = MqttServerClient('test.mosquitto.org', '');

@override
void initState() {
  super.initState();
  setupMqttClient();
}

Future<void> setupMqttClient() async {
  client.logging(on: true);
  client.setProtocolV311();
  client.keepAlivePeriod = 20;
  client.onDisconnected = onDisconnected;
  client.onConnected = onConnected;
  client.onSubscribed = onSubscribed;

  try {
    print('Подключение к MQTT брокеру...');
    await client.connect();
  } catch (e) {
    print('Ошибка подключения - $e');
    client.disconnect();
    return;
  }

  if (client.connectionStatus!.state == MqttConnectionState.connected) {
    print('Подключение установлено!');
  } else {
    print('Подключение не удалось. Статус: ${client.connectionStatus}');
    return;
  }
}

Future<void> getValueFromTopic() async {
  if (client.connectionStatus!.state == MqttConnectionState.connected) {
    print('Подписываемся на топики');
    client.subscribe('IU9/test/a', MqttQos.atLeastOnce);
    client.subscribe('IU9/test/b', MqttQos.atLeastOnce);
    client.subscribe('IU9/test/c', MqttQos.atLeastOnce);

    client.updates!.listen((List<MqttReceivedMessage<MqttMessage?>>? c) {
      final recMess = c![0].payload as MqttPublishMessage;
      final pt = MqttPublishPayload.bytesToStringAsString(recMess.payload.message);

      print('Получено сообщение: topic: ${c[0].topic}, payload: $pt');

      setState(() {
        if (c[0].topic == 'IU9/test/a') {
          _valueA = pt;
        } else if (c[0].topic == 'IU9/test/b') {
          _valueB = pt;
        } else if (c[0].topic == 'IU9/test/c') {
          _valueC = pt;
        }
      });
    });
  } else {
    print('Клиент не подключен к брокеру');
  }
}
```

```
void onSubscribed(String topic) {
  print('Subscription confirmed for topic $topic');
}

void onDisconnected() {
  print('Disconnected from the broker');
}

void onConnected() {
  print('Connected to the broker');
}

@override
Widget build(BuildContext context) {
  return Scaffold(
      appBar: AppBar(
        title: Text('MQTT Subscriber'),
      ),
      body: Container(
      padding: EdgeInsets.all(10.0),
  child: Column(
  crossAxisAlignment: CrossAxisAlignment.start,
  children: <Widget>[
  Text(
  'Значение для топика A:',
  style: TextStyle(fontSize: 20.0),
  ),
  Text(
  _valueA,
  style: TextStyle(fontSize: 18.0),
  ),
  SizedBox(height: 20.0),
  Text(
  'Значение для топика B:',
  style: TextStyle(fontSize: 20.0),
  ),
  Text(
  _valueB,
  style: TextStyle(fontSize: 18.0),
  ),
  SizedBox(height: 20.0),
  Text(
  'Значение для топика C:',
  style: TextStyle(fontSize: 20.0),
  ),
  Text(
  _valueC,
  style: TextStyle(fontSize: 18.0),
  ),
    SizedBox(height: 20.0),
    ElevatedButton(
      child: Text('Получить значения'),
      onPressed: () {
        getValueFromTopic();
        ScaffoldMessenger.of(context).showSnackBar(SnackBar(
          content: Text('Значения обновляются...'),
        ));
      },
      style: ElevatedButton.styleFrom(
          padding: EdgeInsets.symmetric(
              horizontal: 50, vertical: 20),
          textStyle: TextStyle(
              fontSize: 20, fontWeight: FontWeight.bold)),
    ),
```

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

Результаты

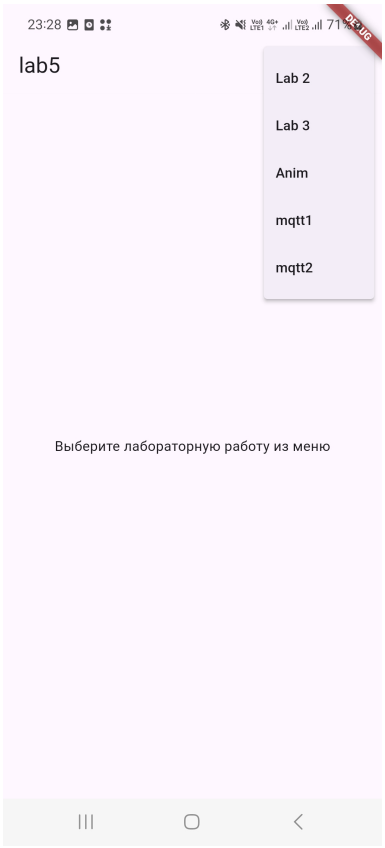

Рис. 1: результаты

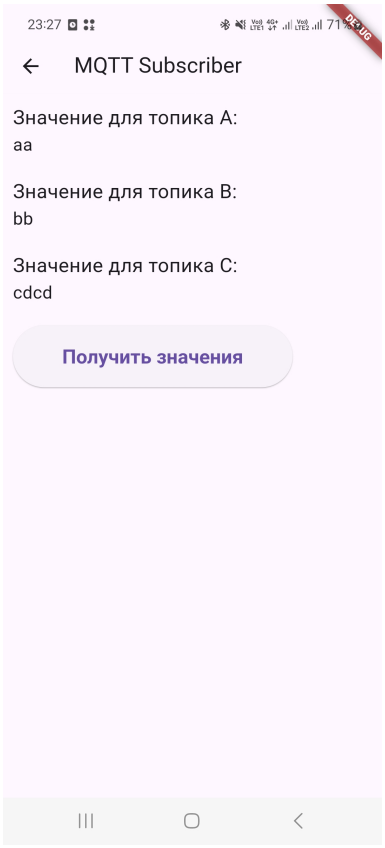

Рис. 2: результаты

Рис. 3: результаты

Вывод

В ходе выполнения лабораторной работы была создана простая система обмена данными по MQTT с помощью двух виджетов для отправки и получения со-

общений. Успешно настроена передача данных из текстовых полей, реализован механизм отображения полученных сообщений и создано выпадающее меню для удобной навигации между виджетами в AppBar.