Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное 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»

Летучка № 3

по курсу «Разработка мобильных приложений»

«Мобильное приложение добавления и удаления записей»

Студент группы ИУ9-72Б Шемякин В.А.

Преподаватель Посевин Д. П.

Москва 2025

1 Задача

Необходимо реализовать мобильное приложение добавления и удаления записей из таблицы базы данных MySQL.

2 Практическая реализация

Код представлен в Листинге 1.

Листинг 1 - main.dart

```
import 'package:flutter/material.dart';
import 'package:http/http.dart' as http;
import 'dart:convert';
import 'dart:math' as math;
import 'package:shared_preferences/shared_preferences.dart';
import 'package:mysql1/mysql1.dart' as mysql;
import 'package:path/path.dart' as path;

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

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Lab     Widget Selector',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.blue),
        useMaterial3: true,
      ),
      home: const WidgetSelectorPage(),
      debugShowCheckedModeBanner: false,
    );
  }
}

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

  void _open(BuildContext context, Widget page) {
    Navigator.of(context).push(MaterialPageRoute(builder: (_) => page));
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('                    ')),
      body: ListView(
      children: [
        ListTile(
          title: const Text('    1    '),
          subtitle: const Text(''),
          trailing: const Icon(Icons.chevron_right),
          onTap: () => _open(context, const Lab1Page()),
        ),
        const Divider(height: 1),

        //            :           3

        ListTile(
          title: const Text('    2    '),
          subtitle: const Text(''),
          trailing: const Icon(Icons.chevron_right),
          onTap: () => _open(context, const Lab2WheelControlPage()),
        ),
        const Divider(height: 1),

        //                  :    4    ->    4
        ListTile(
          title: const Text('    4             '),
          subtitle: const Text(''),
          trailing: const Icon(Icons.chevron_right),
          onTap: () => _open(context, const Lab4Page()),
        ),
        const Divider(height: 1),

        //                :    4         (            )
        ListTile(
          title: const Text('    4         '),
          subtitle: const Text(''),
          trailing: const Icon(Icons.chevron_right),
          onTap: () => _open(context, const Lab4DemoPage()),
        ),
        const Divider(height: 1),
        //                :              3 (MySQL)
        ListTile(
          title: const Text('              3 (MySQL)'),
          subtitle: const Text('         +               +
                    '),
          trailing: const Icon(Icons.chevron_right),
          onTap: () => _open(context, const Letuchka3ConfigPage()),
        ),
        const Divider(height: 1),
          ],
        ),
      );
    }
}

class Lab1Page extends StatefulWidget {
  const Lab1Page({super.key});

  @override
  State<Lab1Page> createState() => _Lab1PageState();
}

class _Lab1PageState extends State<Lab1Page> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() => _counter++);
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Lab1     Flutter Counter')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text('You have pushed the button this many times:'),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headlineMedium,
            ),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}

class Lab2WheelControlPage extends StatefulWidget {
  const Lab2WheelControlPage({super.key});

  @override
  State<Lab2WheelControlPage> createState() => _Lab2WheelControlPageState();
}

class _Lab2WheelControlPageState extends State<Lab2WheelControlPage> {
  int _leftValue = 0;
  int _rightValue = 0;
  bool _serviceEnabled = true;
  bool _isLoading = false;
  String _lastUpdate = '';
  String _directionStatus = '';

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

  void _checkServiceStatus() async {
    try {
      final uri =
          Uri.parse('http://iocontrol.ru/api/readData/BoardVenya2/TestVar2');
      final response = await http.get(uri);
      if (response.statusCode == 200) {
        final jsonResponse = json.decode(response.body);
        final bool check = jsonResponse['check'] ?? false;

        if (check) {
          final statusValue = jsonResponse['value'] ?? "0";
          setState(() {
            _serviceEnabled = statusValue == "1";
          });

          if (_serviceEnabled) {
            _loadWheelsValues();
          }
        }
      }
    } catch (error) {
      debugPrint("Error checking service status: $error");
    }
  }

  void _loadWheelsValues() async {
```

```
  setState(() => _isLoading = true);
  try {
    // left
    final leftUri =
        Uri.parse('http://iocontrol.ru/api/readData/BoardVenya2/left');
    final leftResponse = await http.get(leftUri);
    if (leftResponse.statusCode == 200) {
      final jsonResponse = json.decode(leftResponse.body);
      final bool check = jsonResponse['check'] ?? false;
      if (check) {
        final leftValue = int.tryParse(jsonResponse['value'] ?? '0') ?? 0;
        setState(() => _leftValue = leftValue);
      }
    }

    // right
    final rightUri =
        Uri.parse('http://iocontrol.ru/api/readData/BoardVenya2/right');
    final rightResponse = await http.get(rightUri);
    if (rightResponse.statusCode == 200) {
      final jsonResponse = json.decode(rightResponse.body);
      final bool check = jsonResponse['check'] ?? false;
      if (check) {
        final rightValue = int.tryParse(jsonResponse['value'] ?? '0') ?? 0;
        setState(() => _rightValue = rightValue);
      }
    }
  } catch (error) {
    debugPrint("Error loading wheels values: $error");
  } finally {
    setState(() => _isLoading = false);
  }
}

void _incrementLeft() {
  if (!_serviceEnabled) return;
  setState(() => _leftValue++);
  _updateValuesOnServer();
}

void _decrementLeft() {
  if (!_serviceEnabled) return;
  setState(() => _leftValue--);
  _updateValuesOnServer();
}
```

```
  void _incrementRight() {
    if (!_serviceEnabled) return;
    setState(() => _rightValue++);
    _updateValuesOnServer();
  }

  void _decrementRight() {
    if (!_serviceEnabled) return;
    setState(() => _rightValue--);
    _updateValuesOnServer();
  }

  void _resetValues() {
    if (!_serviceEnabled) return;
    setState(() {
      _leftValue = 0;
      _rightValue = 0;
      _directionStatus = '';
    });
    _updateValuesOnServer();
  }

  void _updateValuesOnServer() async {
    try {
      final leftUri = Uri.parse(
          'http://iocontrol.ru/api/sendData/BoardVenya2/left/$_leftValue');
      final rightUri = Uri.parse(
          'http://iocontrol.ru/api/sendData/BoardVenya2/right/$_rightValue');

      final leftResponse = await http.get(leftUri);
      final rightResponse = await http.get(rightUri);

      if (leftResponse.statusCode == 200 && rightResponse.statusCode == 200)
         {
        setState(() => _lastUpdate = DateTime.now().toString());
      }
    } catch (error) {
      debugPrint("Error updating values: $error");
    }
  }

  void _getDeviceStatus() {
    if (!_serviceEnabled) return;
    setState(() {
      if (_rightValue > _leftValue) {
        _directionStatus = '                              ';
      } else if (_leftValue > _rightValue) {
        _directionStatus = '                              ';
      } else if (_leftValue == _rightValue && _leftValue == 0) {
        _directionStatus = '                                     ';
      } else {
        _directionStatus = '                              ';
      }
    });
  }

  void _getBitovkaLampRequestON() {
    setState(() => _isLoading = true);
    final uri =
        Uri.parse('http://iocontrol.ru/api/sendData/BoardVenya2/TestVar2/1');
    http.get(uri).then((response) {
      setState(() => _serviceEnabled = true);
      _loadWheelsValues();
    }).catchError((error) {
      setState(() => _isLoading = false);
      debugPrint("Error turning on service: $error");
    });
  }

  void _getBitovkaLampRequestOFF() {
    setState(() => _isLoading = true);
    final uri =
        Uri.parse('http://iocontrol.ru/api/sendData/BoardVenya2/TestVar2/0');
    http.get(uri).then((response) {
      setState(() {
        _serviceEnabled = false;
        _isLoading = false;
      });
    }).catchError((error) {
      setState(() => _isLoading = false);
      debugPrint("Error turning off service: $error");
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Lab2     Wheel Control'),
        actions: [
          IconButton(
            icon: const Icon(Icons.refresh),
            onPressed: _serviceEnabled ? _loadWheelsValues : null,
            tooltip: 'Refresh values',
          ),
        ],
      ),
      body: Center(
        child: _isLoading
            ? const Column(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  CircularProgressIndicator(),
                  SizedBox(height: 20),
                  Text('Loading...'),
                ],
              )
            : SingleChildScrollView(
                padding: const EdgeInsets.all(16.0),
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: <Widget>[
                    // Service Status
                    Card(
                      elevation: 4,
                      child: Padding(
                        padding: const EdgeInsets.all(16.0),
                        child: Column(
                          children: [
                            Text('Service Status:',
                                style:
                                    Theme.of(context).textTheme.titleMedium),
                            const SizedBox(height: 8),
                            Text(
                              _serviceEnabled ? 'ENABLED' : 'DISABLED',
                              style: TextStyle(
                                color:
                                    _serviceEnabled ? Colors.green : Colors.
                                        red,
                                fontWeight: FontWeight.bold,
                                fontSize: 20,
                              ),
                            ),
                          ],
                        ),
                      ),
                    ),
            const SizedBox(height: 20),

            // Left Wheel
            Card(
              elevation: 4,
              child: Padding(
                padding: const EdgeInsets.all(16.0),
                child: Column(
                  children: [
                    Text('Left Wheel:',
                        style:
                            Theme.of(context).textTheme.titleMedium),
                    const SizedBox(height: 8),
                    Text('$_leftValue',
                        style: const TextStyle(
                          fontSize: 32,
                          fontWeight: FontWeight.bold,
                        )),
                    const SizedBox(height: 12),
                    Row(
                      mainAxisAlignment: MainAxisAlignment.center,
                      children: [
                        ElevatedButton(
                          style: ElevatedButton.styleFrom(
                            backgroundColor: Colors.red,
                            foregroundColor: Colors.white,
                            minimumSize: const Size(60, 60),
                            shape: const CircleBorder(),
                          ),
                          onPressed:
                              _serviceEnabled ? _decrementLeft : null
                                  ,
                          child: const Icon(Icons.remove, size: 30),
                        ),
                        const SizedBox(width: 20),
                        ElevatedButton(
                          style: ElevatedButton.styleFrom(
                            backgroundColor: Colors.green,
                            foregroundColor: Colors.white,
                            minimumSize: const Size(60, 60),
                            shape: const CircleBorder(),
                          ),
                          onPressed:
                              _serviceEnabled ? _incrementLeft : null
                                  ,
                          child: const Icon(Icons.add, size: 30),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),

      const SizedBox(height: 20),

      // Right Wheel
      Card(
        elevation: 4,
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              Text('Right Wheel:',
                  style:
                      Theme.of(context).textTheme.titleMedium),
              const SizedBox(height: 8),
              Text('$_rightValue',
                  style: const TextStyle(
                    fontSize: 32,
                    fontWeight: FontWeight.bold,
                  )),
              const SizedBox(height: 12),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.red,
                      foregroundColor: Colors.white,
                      minimumSize: const Size(60, 60),
                      shape: const CircleBorder(),
                    ),
                    onPressed:
                        _serviceEnabled ? _decrementRight :
                            null,
                    child: const Icon(Icons.remove, size: 30),
                  ),
                  const SizedBox(width: 20),
                  ElevatedButton(
                    style: ElevatedButton.styleFrom(
                      backgroundColor: Colors.green,
                      foregroundColor: Colors.white,
                      minimumSize: const Size(60, 60),
                      shape: const CircleBorder(),
                    ),
                    onPressed:
                        _serviceEnabled ? _incrementRight :
                            null,
                    child: const Icon(Icons.add, size: 30),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),

      const SizedBox(height: 20),

      // Device Status
      Card(
        elevation: 4,
        child: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              Text('Device Status:',
                  style:
                      Theme.of(context).textTheme.titleMedium),
              const SizedBox(height: 8),
              Text(
                _directionStatus.isEmpty
                    ? 'Press "Get Status" to check'
                    : _directionStatus,
                style: TextStyle(
                  color: _directionStatus.isEmpty
                      ? Colors.grey
                      : Colors.blue,
                  fontWeight: FontWeight.bold,
                  fontSize: 18,
                ),
              ),
              const SizedBox(height: 12),
              ElevatedButton(
                style: ElevatedButton.styleFrom(
                  backgroundColor: Colors.purple,
                  foregroundColor: Colors.white,
              minimumSize: const Size(200, 50),
            ),
            onPressed:
                _serviceEnabled ? _getDeviceStatus : null,
            child: const Text('GET DEVICE STATUS'),
          ),
        ],
      ),
    ),
),

const SizedBox(height: 20),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.blue,
        foregroundColor: Colors.white,
      ),
      onPressed: _getBitovkaLampRequestON,
      child: const Text('TURN ON SERVICE'),
    ),
    const SizedBox(width: 10),
    ElevatedButton(
      style: ElevatedButton.styleFrom(
        backgroundColor: Colors.grey,
        foregroundColor: Colors.white,
      ),
      onPressed: _getBitovkaLampRequestOFF,
      child: const Text('TURN OFF SERVICE'),
    ),
  ],
),

const SizedBox(height: 20),

ElevatedButton(
  style: ElevatedButton.styleFrom(
    backgroundColor: Colors.orange,
    foregroundColor: Colors.white,
    minimumSize: const Size(200, 50),
  ),
  onPressed: _serviceEnabled ? _resetValues : null,
  child: const Text('RESET VALUES TO 0'),
),
                    const SizedBox(height: 10),
                    Text(
                      'Range: -100 to 100',
                      style: TextStyle(
                        color: Colors.grey[600],
                        fontStyle: FontStyle.italic,
                      ),
                    ),

                    if (_lastUpdate.isNotEmpty) ...[
                      const SizedBox(height: 10),
                      Text(
                        'Last update: $_lastUpdate',
                        style: TextStyle(
                          color: Colors.grey[600],
                          fontSize: 12,
                        ),
                      ),
                    ],
                  ],
                ),
              ),
      ),
    );
  }
}

class EmptyPage extends StatelessWidget {
  final String title;
  final String subtitle;
  const EmptyPage({super.key, required this.title, required this.subtitle});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('$title               ')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const Icon(Icons.info_outline, size: 64),
            const SizedBox(height: 12),
            Text(
              subtitle,
              textAlign: TextAlign.center,
              style: Theme.of(context).textTheme.titleMedium,
            ),
          ],
        ),
      ),
    );
  }
}

class Lab4Page extends StatefulWidget {
  const Lab4Page({super.key});

  @override
  State<Lab4Page> createState() => _Lab4PageState();
}

class _Lab4PageState extends State<Lab4Page> {
  static const double minLen = 10;
  static const double maxLen = 200;

  static const _kA = 'lab4_a';
  static const _kB = 'lab4_b';
  static const _kC = 'lab4_c';
  static const _kHidden = 'lab4_showHidden';

  double a = 120;
  double b = 80;
  double c = 100;
  bool showHidden = true;

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

  Future<void> _loadSettings() async {
    final sp = await SharedPreferences.getInstance();
    setState(() {
      a = sp.getDouble(_kA) ?? 120;
      b = sp.getDouble(_kB) ?? 80;
      c = sp.getDouble(_kC) ?? 100;
      showHidden = sp.getBool(_kHidden) ?? true;
    });
  }
```

```
Future<void> _saveSettings() async {
  final sp = await SharedPreferences.getInstance();
  await sp.setDouble(_kA, a);
  await sp.setDouble(_kB, b);
  await sp.setDouble(_kC, c);
  await sp.setBool(_kHidden, showHidden);
}

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

@override
Widget build(BuildContext context) {
  return Scaffold(
    //
    appBar: AppBar(title: const Text('   4                  ')),
    body: Column(
      children: [
        Card(
          margin: const EdgeInsets.all(12),
          child: Padding(
            padding: const EdgeInsets.symmetric(horizontal: 12, vertical:
                8),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                _lenSlider(
                  label: 'a',
                  value: a,
                  onChanged: (v) => setState(() { a = v; }),
                  onChangeEnd: (_) => _saveSettings(),
                ),
                _lenSlider(
                  label: 'b',
                  value: b,
                  onChanged: (v) => setState(() { b = v; }),
                  onChangeEnd: (_) => _saveSettings(),
                ),
                _lenSlider(
                  label: 'c',
                  value: c,
                  onChanged: (v) => setState(() { c = v; }),
                  onChangeEnd: (_) => _saveSettings(),
        ),
        const SizedBox(height: 4),
        Row(
          children: [
            FilterChip(
              label: const Text('
                                              '),
              selected: showHidden,
              onSelected: (v) {
                setState(() => showHidden = v);
                _saveSettings();
              },
            ),
            const Spacer(),
            TextButton.icon(
              onPressed: () {
                setState(() {
                  a = 120; b = 80; c = 100; showHidden = true;
                });
                _saveSettings();
              },
              icon: const Icon(Icons.restart_alt),
              label: const Text('          '),
            ),
          ],
        ),
      ],
    ),
  ),
),
Expanded(
  child: Center(
    child: AspectRatio(
      aspectRatio: 1.2,
      child: Card(
        margin: const EdgeInsets.all(12),
        child: Padding(
          padding: const EdgeInsets.all(8.0),
          child: CustomPaint(
            painter: IsoParallelepipedPainter(
              a,
              b,
              c,
              showHidden: showHidden,
            ),
            willChange: true,
                  ),
                ),
              ),
            ),
          ),
        ),
      ],
    ),
  );
}

Widget _lenSlider({
  required String label,
  required double value,
  required ValueChanged<double> onChanged,
  ValueChanged<double>? onChangeEnd,
}) {
  const double minLen = _Lab4PageState.minLen;
  const double maxLen = _Lab4PageState.maxLen;

  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
          Text('$label:', style: const TextStyle(fontWeight: FontWeight.
             w600)),
          const SizedBox(width: 8),
          Text(value.toStringAsFixed(0)),
          const Spacer(),
          Text('${minLen.toStringAsFixed(0)}     ${maxLen.toStringAsFixed
             (0)}'),
        ],
      ),
      Slider.adaptive(
        value: value.clamp(minLen, maxLen).toDouble(),
        min: minLen,
        max: maxLen,
        divisions: (maxLen - minLen).toInt(),
        label: value.toStringAsFixed(0),
        onChanged: onChanged,
        onChangeEnd: onChangeEnd,
      ),
      const SizedBox(height: 4),
    ],
  );
  }
}

class IsoParallelepipedPainter extends CustomPainter {
  final double a, b, c;
  final bool showHidden;

  IsoParallelepipedPainter(this.a, this.b, this.c, {required this.showHidden
     });

  static const double _cos30 = 0.8660254037844386;
  static const double _sin30 = 0.5;

  Offset _iso(double x, double y, double z) {
    final sx = (x - y) * _cos30;
    final sy = (x + y) * _sin30 - z;
    return Offset(sx, sy);
  }

  List<(String, String)> get _edges => const [
        // X-
        ('000', '100'),
        ('010', '110'),
        ('001', '101'),
        ('011', '111'),
        // Y-
        ('000', '010'),
        ('100', '110'),
        ('001', '011'),
        ('101', '111'),
        // Z-
        ('000', '001'),
        ('100', '101'),
        ('010', '011'),
        ('110', '111'),
      ];

  Set<(String, String)> get _visibleEdges {
    final s = <(String, String)>{};

    void addFaceEdges(List<String> vs) {
      final e = <(String, String)>[
        (vs[0], vs[1]),
        (vs[1], vs[3]),
        (vs[3], vs[2]),
        (vs[2], vs[0]),
    ];
    for (final edge in e) {
      final sorted = _sortEdge(edge);
      s.add(sorted);
    }
  }

  addFaceEdges(['000', '100', '001', '101']); // y = 0 (                  )
  addFaceEdges(['100', '110', '101', '111']); // x = a (              )
  addFaceEdges(['001', '101', '011', '111']); // z = c (                )

  return s;
}

(String, String) _sortEdge((String, String) e) {
  final a = e.$1;
  final b = e.$2;
  return (a.compareTo(b) <= 0) ? (a, b) : (b, a);
}

@override
void paint(Canvas canvas, Size size) {
  if (a <= 0 || b <= 0 || c <= 0) {
    _drawCenteredText(canvas, size, '                                > 0')
      ;
    return;
  }

  final Map<String, Offset> p2d = {};

  const margin = 24.0;
  final w = size.width - 2 * margin;
  final h = size.height - 2 * margin;

  final pts = <Offset>[
    _iso(0, 0, 0),
    _iso(a, 0, 0),
    _iso(0, b, 0),
    _iso(a, b, 0),
    _iso(0, 0, c),
    _iso(a, 0, c),
    _iso(0, b, c),
    _iso(a, b, c),
  ];

  final bounds = _pointsBounds(pts);
final sx = w / bounds.width;
final sy = h / bounds.height;
final scale = 0.9 * math.min(sx, sy);

final center = Offset(size.width / 2, size.height / 2);
final geoCenter = Offset(bounds.left + bounds.width / 2, bounds.top +
    bounds.height / 2);

void put(String key, double x, double y, double z) {
  final p = _iso(x, y, z);
  final q = (p - geoCenter) * scale + center;
  p2d[key] = q;
}

// 8
put('000', 0, 0, 0);
put('100', a, 0, 0);
put('010', 0, b, 0);
put('110', a, b, 0);
put('001', 0, 0, c);
put('101', a, 0, c);
put('011', 0, b, c);
put('111', a, b, c);

Offset? v(String k) => p2d[k];

final paintSolid = Paint()
  ..style = PaintingStyle.stroke
  ..strokeWidth = 2.0
  ..color = Colors.black;

final paintHidden = Paint()
  ..style = PaintingStyle.stroke
  ..strokeWidth = 1.5
  ..color = Colors.grey;

final paintFill = Paint()
  ..style = PaintingStyle.fill
  ..color = const Color(0x99FFFFFF); //


final visible = _visibleEdges;
final all = _edges.map(_sortEdge).toSet();
final hidden = all.difference(visible);

if (showHidden) {
      for (final e in hidden) {
        final p1 = v(e.$1), p2 = v(e.$2);
        if (p1 != null && p2 != null) {
          _drawDashedLine(canvas, p1, p2, paintHidden, dash: 8, gap: 6);
        }
      }
    }

    Path face(List<String> vs) {
      final a = v(vs[0]), b = v(vs[1]), d = v(vs[3]), c = v(vs[2]);
      final path = Path();
      if (a == null || b == null || c == null || d == null) return path;
      path..moveTo(a.dx, a.dy)..lineTo(b.dx, b.dy)..lineTo(d.dx, d.dy)..
          lineTo(c.dx, c.dy)..close();
      return path;
    }

    for (final f in [
      ['000', '100', '001', '101'], // y=0
      ['100', '110', '101', '111'], // x=a
      ['001', '101', '011', '111'], // z=c
    ]) {
      final path = face(f);
      if (path.computeMetrics().isNotEmpty) {
        canvas.drawPath(path, paintFill);
      }
    }

    for (final e in visible) {
      final p1 = v(e.$1), p2 = v(e.$2);
      if (p1 != null && p2 != null) {
        canvas.drawLine(p1, p2, paintSolid);
      }
    }
  }

  @override
  bool shouldRepaint(covariant IsoParallelepipedPainter old) {
    return a != old.a || b != old.b || c != old.c || showHidden != old.
        showHidden;
  }

  Rect _pointsBounds(List<Offset> pts) {
    double minX = double.infinity, minY = double.infinity;
    double maxX = -double.infinity, maxY = -double.infinity;
    for (final p in pts) {
      if (p.dx < minX) minX = p.dx;
      if (p.dy < minY) minY = p.dy;
      if (p.dx > maxX) maxX = p.dx;
      if (p.dy > maxY) maxY = p.dy;
    }
    return Rect.fromLTRB(minX, minY, maxX, maxY);
  }

  void _drawDashedLine(Canvas canvas, Offset a, Offset b, Paint paint,
      {double dash = 6, double gap = 4}) {
    final total = (b - a).distance;
    final dir = (b - a) / total;
    double t = 0;
    while (t < total) {
      final tNext = math.min(t + dash, total);
      final p1 = a + dir * t;
      final p2 = a + dir * tNext;
      canvas.drawLine(p1, p2, paint);
      t = tNext + gap;
    }
  }

  void _drawCenteredText(Canvas canvas, Size size, String text) {
    final tp = TextPainter(
      text: const TextSpan(
        text: '                              > 0',
        style: TextStyle(fontSize: 16, color: Colors.grey),
      ),
      textDirection: TextDirection.ltr,
    )..layout(maxWidth: size.width - 40);
    final pos = Offset(
      (size.width - tp.width) / 2,
      (size.height - tp.height) / 2,
    );
    tp.paint(canvas, pos);
  }
}

class Lab4DemoPage extends StatefulWidget {
  const Lab4DemoPage({super.key});

  @override
  State<Lab4DemoPage> createState() => _Lab4DemoPageState();
}

class _Lab4DemoPageState extends State<Lab4DemoPage> {
```

```
double _sides = 3.0;
double _radius = 100.0;
double _radians = 0.0;
// ===                                               _Lab4DemoPageState
    ===
static const _kSides   = 'lab4demo_sides';
static const _kRadius  = 'lab4demo_radius';
static const _kRadians = 'lab4demo_radians';

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

Future<void> _loadDemoPrefs() async {
  final sp = await SharedPreferences.getInstance();
  setState(() {
    _sides   = sp.getDouble(_kSides)   ?? 3.0;
    _radius  = sp.getDouble(_kRadius)  ?? 100.0;
    _radians = sp.getDouble(_kRadians) ?? 0.0;
  });
}

Future<void> _saveDemoPrefs() async {
  final sp = await SharedPreferences.getInstance();
  await sp.setDouble(_kSides, _sides);
  await sp.setDouble(_kRadius, _radius);
  await sp.setDouble(_kRadians, _radians);
}

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

@override
Widget build(BuildContext context) {
  final maxR = MediaQuery.of(context).size.width / 2;

  return Scaffold(
    appBar: AppBar(
      title: const Text('   4                              '),
    ),
    body: SafeArea(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            Expanded(
              child: CustomPaint(
                painter: ShapePainter(_sides, _radius, _radians),
                child: const SizedBox.expand(),
              ),
            ),
            const Padding(
              padding: EdgeInsets.only(left: 16.0, top: 8),
              child: Text('Sides'),
            ),
            // Sides
            Slider(
              value: _sides,
              min: 3.0,
              max: 10.0,
              label: _sides.toInt().toString(),
              divisions: 7,
              onChanged: (value) => setState(() => _sides = value),
              onChangeEnd: (_) => _saveDemoPrefs(),
            ),
            Padding(
              padding: const EdgeInsets.only(left: 16.0),
              child: Text('Size'),
            ),
            // Size
            Slider(
              value: _radius.clamp(10.0, maxR),
              min: 10.0,
              max: maxR,
              onChanged: (value) => setState(() => _radius = value),
              onChangeEnd: (_) => _saveDemoPrefs(),
            ),
            Padding(
              padding: const EdgeInsets.only(left: 16.0),
              child: Text('Rotation'),
            ),
            // Rotation
            Slider(
              value: _radians,
              min: 0.0,
              max: math.pi,
              onChanged: (value) => setState(() => _radians = value),
              onChangeEnd: (_) => _saveDemoPrefs(),
              ),

            ],
          ),
        ),
      );
    }
  }

// Painter
class ShapePainter extends CustomPainter {
  final double sides;
  final double radius;
  final double radians;
  ShapePainter(this.sides, this.radius, this.radians);

  @override
  void paint(Canvas canvas, Size size) {
    final paint = Paint()
      ..color = Colors.teal
      ..strokeWidth = 5
      ..style = PaintingStyle.stroke
      ..strokeCap = StrokeCap.round;

    final path = Path();
    final angle = (math.pi * 2) / sides;

    final center = Offset(size.width / 2, size.height / 2);
    final startPoint = Offset(
      radius * math.cos(radians),
      radius * math.sin(radians),
    );

    path.moveTo(startPoint.dx + center.dx, startPoint.dy + center.dy);

    for (int i = 1; i <= sides; i++) {
      final x = radius * math.cos(radians + angle * i) + center.dx;
      final y = radius * math.sin(radians + angle * i) + center.dy;
      path.lineTo(x, y);
    }
    path.close();
    canvas.drawPath(path, paint);
  }

  @override
  bool shouldRepaint(covariant CustomPainter oldDelegate) => true;
```

```
}

/// ======                                    ,                                    ======
const String kDefaultDbUser = 'iu9mobile';
const String kDefaultDbPass = 'bmstubmstu123';
const String kDefaultDbName = 'iu9mobile';

/// ====================== MySQL SERVICE ======================
class MySqlService {
  mysql.MySqlConnection? _conn;
  final String table; //                          ,                          :
      Shemyakin_Mobile

  MySqlService(this.table);

  Future<void> connect({
    required String host,
    required String db,
    required String user,
    required String password,
  }) async {
    await close();
    final settings = mysql.ConnectionSettings(
      host: host,
      user: user,
      password: password,
      db: db,
    );
    _conn = await mysql.MySqlConnection.connect(settings);
    await _ensureTable();
  }

  bool get isOpen => _conn != null;

  Future<void> _ensureTable() async {
    final sql = '''
      CREATE TABLE IF NOT EXISTS `$table` (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(100) NOT NULL,
        email VARCHAR(150) NOT NULL,
        age INT NOT NULL
      ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
    ''';
    await _conn!.query(sql);
  }
  Future<void> insertPerson({
    required String name,
    required String email,
    required int age,
  }) async {
    final sql = 'INSERT INTO `$table` (name, email, age) VALUES (?, ?, ?)';
    await _conn!.query(sql, [name, email, age]);
  }

  Future<List<Map<String, dynamic>>> fetchAll() async {
    final results = await _conn!
        .query('SELECT id, name, email, age FROM `$table` ORDER BY id DESC');
    return results
        .map((r) => {
              'id': r[0],
              'name': r[1],
              'email': r[2],
              'age': r[3],
            })
        .toList();
  }

  Future<void> deleteById(int id) async {
    await _conn!.query('DELETE FROM `$table` WHERE id = ?', [id]);
  }

  Future<void> close() async {
    await _conn?.close();
    _conn = null;
  }
}


/// ======================            1:            ======================
class Letuchka3ConfigPage extends StatefulWidget {
  const Letuchka3ConfigPage({super.key});

  @override
  State<Letuchka3ConfigPage> createState() => _Letuchka3ConfigPageState();
}

class _Letuchka3ConfigPageState extends State<Letuchka3ConfigPage> {
  final _form = GlobalKey<FormState>();
  final _hostCtrl = TextEditingController();
  final _dbCtrl = TextEditingController(text: kDefaultDbName);
  final _loginCtrl = TextEditingController(text: kDefaultDbUser);
  final _passCtrl = TextEditingController(text: kDefaultDbPass);
  final _surnameCtrl = TextEditingController();

  bool _busy = false;
  String? _error;

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

  Future<void> _loadPrefs() async {
    final sp = await SharedPreferences.getInstance();
    _hostCtrl.text = sp.getString('l3_host') ?? '';
    _dbCtrl.text = sp.getString('l3_db') ?? kDefaultDbName;
    _loginCtrl.text = sp.getString('l3_login') ?? kDefaultDbUser;
    _passCtrl.text = sp.getString('l3_pass') ?? kDefaultDbPass;
    _surnameCtrl.text = sp.getString('l3_surname') ?? '';
    setState(() {});
  }

  Future<void> _savePrefs() async {
    final sp = await SharedPreferences.getInstance();
    await sp.setString('l3_host', _hostCtrl.text.trim());
    await sp.setString(
        'l3_db',
        _dbCtrl.text.trim().isEmpty
            ? kDefaultDbName
            : _dbCtrl.text.trim());
    await sp.setString(
        'l3_login',
        _loginCtrl.text.trim().isEmpty
            ? kDefaultDbUser
            : _loginCtrl.text.trim());
    await sp.setString(
        'l3_pass', _passCtrl.text.isEmpty ? kDefaultDbPass : _passCtrl.text);
    await sp.setString('l3_surname', _surnameCtrl.text.trim());
  }

  @override
  void dispose() {
    _hostCtrl.dispose();
    _dbCtrl.dispose();
    _loginCtrl.dispose();
    _passCtrl.dispose();
    _surnameCtrl.dispose();
    super.dispose();
  }

  Future<void> _onOk() async {
    if (!_form.currentState!.validate()) return;
    setState(() {
      _busy = true;
      _error = null;
    });

    try {
      await _savePrefs();
      if (!mounted) return;
      Navigator.of(context).push(MaterialPageRoute(
        builder: (_) => Letuchka3FormPage(
          host: _hostCtrl.text.trim(),
          db: _dbCtrl.text.trim(),
          user: _loginCtrl.text.trim(),
          password: _passCtrl.text,
          surname: _surnameCtrl.text.trim(),
        ),
      ));
    } catch (e) {
      _error = '$e';
    } finally {
      if (mounted) setState(() => _busy = false);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('              3                   (
         MySQL)')),
      body: SafeArea(
        child: AbsorbPointer(
          absorbing: _busy,
          child: SingleChildScrollView(
            padding: const EdgeInsets.all(16),
            child: Form(
              key: _form,
              child: Column(
                children: [
                  TextFormField(
                    controller: _hostCtrl,
    decoration: const InputDecoration(
      labelText: 'host (ip/          :          )',
    ),
    validator: (v) =>
        (v == null || v.trim().isEmpty) ? '
           host' : null,
  ),
  TextFormField(
    controller: _dbCtrl,
    decoration: const InputDecoration(labelText: 'db (
               )'),
    validator: (v) =>
        (v == null || v.trim().isEmpty) ? '
                      ' : null,
  ),
  TextFormField(
    controller: _loginCtrl,
    decoration: const InputDecoration(labelText: 'login (user
       )'),
    validator: (v) =>
        (v == null || v.trim().isEmpty) ? '
                     ' : null,
  ),
  TextFormField(
    controller: _passCtrl,
    decoration: const InputDecoration(labelText: 'password'),
    obscureText: true,
    validator: (v) =>
        (v == null || v.isEmpty) ? '
                       ' : null,
  ),
  TextFormField(
    controller: _surnameCtrl,
    decoration: const InputDecoration(labelText: 'surname'),
    validator: (v) =>
        (v == null || v.trim().isEmpty) ? '
                         ' : null,
  ),
  const SizedBox(height: 16),
  if (_error != null)
    Text(_error!, style: const TextStyle(color: Colors.red)),
  const SizedBox(height: 8),
  ElevatedButton.icon(
    onPressed: _onOk,
    icon: _busy
        ? const SizedBox(
                              width: 16,
                              height: 16,
                              child: CircularProgressIndicator(strokeWidth: 2),
                            )
                          : const Icon(Icons.arrow_forward),
                      label: const Text('     '),
                    ),
                  ],
                ),
              ),
            ),
          ),
        ),
      );
    }
  }

  /// ====================                2:              (Add / Del)
     ====================
  class Letuchka3FormPage extends StatefulWidget {
    final String host, db, user, password, surname;
    const Letuchka3FormPage({
      super.key,
      required this.host,
      required this.db,
      required this.user,
      required this.password,
      required this.surname,
    });

    @override
    State<Letuchka3FormPage> createState() => _Letuchka3FormPageState();
  }

  class _Letuchka3FormPageState extends State<Letuchka3FormPage> {
    final _form = GlobalKey<FormState>();
    final _nameCtrl = TextEditingController();
    final _emailCtrl = TextEditingController();
    final _ageCtrl = TextEditingController();

    late MySqlService _service; // <                  late

    bool _busy = false;
    String? _err;

    String _tableFromSurname(String s) {
      final base = s.trim().isEmpty ? 'Unknown' : s.trim();
      final sanitized = base.replaceAll(RegExp(r'[^a-zA-Z0-9_]'), '_');
      return '${sanitized}_Mobile';
    }

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

    Future<void> _connect() async {
    setState(() { _busy = true; _err = null; });
    try {
      //                                ":          "      host
      var host = widget.host.trim();
      if (host.contains(':')) {
        host = host.split(':').first;
      }

      final tableName = _tableFromSurname(widget.surname);
      _service = MySqlService(tableName);

      await _service.connect(
        host: host,
        db: widget.db,
        user: widget.user,
        password: widget.password,
      );
    } catch (e) {
      _err = '                                          : $e';
    } finally {
      if (mounted) setState(() => _busy = false);
    }
  }


    @override
    void dispose() {
      _nameCtrl.dispose();
      _emailCtrl.dispose();
      _ageCtrl.dispose();
      _service.close();
      super.dispose();
    }
```

```
Future<void> _onAdd() async {
  if (!_service.isOpen) {
    setState(() => _err = '                                    ');
    return;
  }
  if (!_form.currentState!.validate()) return;

  setState(() {
    _busy = true;
    _err = null;
  });
  try {
    await _service.insertPerson(
      name: _nameCtrl.text.trim(),
      email: _emailCtrl.text.trim(),
      age: int.parse(_ageCtrl.text.trim()),
    );
    if (!mounted) return;
    Navigator.of(context).push(MaterialPageRoute(
      builder: (_) => Letuchka3TablePage(service: _service),
    ));
  } catch (e) {
    setState(() => _err = '                                  : $e');
  } finally {
    if (mounted) setState(() => _busy = false);
  }
}

void _onDel() {
  if (!_service.isOpen) {
    setState(() => _err = '                                    ');
    return;
  }
  Navigator.of(context).push(MaterialPageRoute(
    builder: (_) => Letuchka3DeletePage(service: _service),
  ));
}

@override
Widget build(BuildContext context) {
  final disabled = _busy || !_service.isOpen;

  return Scaffold(
    appBar: AppBar(title: const Text('              3               ')),
    body: SafeArea(
      child: AbsorbPointer(
absorbing: _busy,
child: SingleChildScrollView(
  padding: const EdgeInsets.all(16),
  child: Column(
    children: [
      if (_busy) const LinearProgressIndicator(),
      if (_err != null) ...[
        const SizedBox(height: 8),
        Text(_err!, style: const TextStyle(color: Colors.red)),
      ],
      const SizedBox(height: 8),
      Form(
        key: _form,
        child: Column(
          children: [
            TextFormField(
              controller: _nameCtrl,
              decoration: const InputDecoration(labelText: 'name'),
              validator: (v) =>
                  (v == null || v.trim().isEmpty) ? '
                            ' : null,
            ),
            TextFormField(
              controller: _emailCtrl,
              decoration: const InputDecoration(labelText: 'email')
                 ,
              keyboardType: TextInputType.emailAddress,
              validator: (v) {
                if (v == null || v.trim().isEmpty) {
                  return '               email';
                }
                if (!v.contains('@')) return '
                                            email';
                return null;
              },
            ),
            TextFormField(
              controller: _ageCtrl,
              decoration: const InputDecoration(labelText: 'age'),
              keyboardType: TextInputType.number,
              validator: (v) {
                final n = int.tryParse(v ?? '');
                if (n == null || n < 0) return '
                                                      ';
                return null;
              },
                      ),
                    ],
                  ),
                ),
                const SizedBox(height: 16),
                Row(
                  children: [
                    Expanded(
                      child: ElevatedButton(
                        onPressed: disabled ? null : _onAdd,
                        child: const Text('Add'),
                      ),
                    ),
                    const SizedBox(width: 12),
                    Expanded(
                      child: OutlinedButton(
                        onPressed: disabled ? null : _onDel,
                        child: const Text('Del'),
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      ),
    );
  }
}

/// ====================          3  :                    + Back
    ====================
class Letuchka3TablePage extends StatefulWidget {
  final MySqlService service;
  const Letuchka3TablePage({super.key, required this.service});

  @override
  State<Letuchka3TablePage> createState() => _Letuchka3TablePageState();
}

class _Letuchka3TablePageState extends State<Letuchka3TablePage> {
  bool _loading = true;
  String? _err;
  List<Map<String, dynamic>> _rows = [];
```

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

Future<void> _load() async {
  setState(() {
    _loading = true;
    _err = null;
  });
  try {
    _rows = await widget.service.fetchAll();
  } catch (e) {
    _err = '                             : $e';
  } finally {
    if (mounted) setState(() => _loading = false);
  }
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: const Text('                    Shemyakin')),
    body: _loading
        ? const Center(child: CircularProgressIndicator())
        : _err != null
            ? Center(child: Text(_err!, style: const TextStyle(color:
               Colors.red)))
            : Column(
                children: [
                  Expanded(
                    child: SingleChildScrollView(
                      scrollDirection: Axis.horizontal,
                      child: DataTable(
                        columns: const [
                          DataColumn(label: Text('ID')),
                          DataColumn(label: Text('Name')),
                          DataColumn(label: Text('Email')),
                          DataColumn(label: Text('Age')),
                        ],
                        rows: _rows.map((r) => DataRow(cells: [
                          DataCell(Text('${r['id']}')),
                          DataCell(Text('${r['name']}')),
                          DataCell(Text('${r['email']}')),
                          DataCell(Text('${r['age']}')),
                          ])).toList(),

                        ),
                      ),
                    ),
                    const SizedBox(height: 8),
                    Padding(
                      padding: const EdgeInsets.all(12.0),
                      child: ElevatedButton.icon(
                        onPressed: () => Navigator.of(context).pop(),
                        icon: const Icon(Icons.arrow_back),
                        label: const Text('Back'),
                      ),
                    ),
                  ],
                ),
    );
  }
}

/// =====================            3   :                  + Back
    =====================
class Letuchka3DeletePage extends StatefulWidget {
  final MySqlService service;
  const Letuchka3DeletePage({super.key, required this.service});

  @override
  State<Letuchka3DeletePage> createState() => _Letuchka3DeletePageState();
}

class _Letuchka3DeletePageState extends State<Letuchka3DeletePage> {
  bool _loading = true;
  String? _err;
  List<Map<String, dynamic>> _rows = [];

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

  Future<void> _load() async {
    setState(() {
      _loading = true;
      _err = null;
    });
    try {
      _rows = await widget.service.fetchAll();
    } catch (e) {
      _err = '                              : $e';
    } finally {
      if (mounted) setState(() => _loading = false);
    }
  }

  Future<void> _delete(int id) async {
    setState(() {
      _loading = true;
      _err = null;
    });
    try {
      await widget.service.deleteById(id);
      await _load(); //
    } catch (e) {
      setState(() {
        _err = '                              : $e';
        _loading = false;
      });
    }
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('                                ')),
      body: _loading
          ? const Center(child: CircularProgressIndicator())
          : _err != null
              ? Center(child: Text(_err!, style: const TextStyle(color:
                 Colors.red)))
              : Column(
                  children: [
                    Expanded(
                      child: ListView.separated(
                        itemCount: _rows.length,
                        separatorBuilder: (_, __) => const Divider(height: 1)
                           ,
                        itemBuilder: (context, i) {
                          final r = _rows[i];
                          return ListTile(
                            title: Text('${r['name']}     ${r['email']}'),
                            subtitle: Text(
                                'ID: ${r['id']} | age: ${r['age']} | surname:
                                    ${r['surname']}'),
                            trailing: IconButton(
                              icon: const Icon(Icons.delete, color: Colors.
                                  red),
                              onPressed: () => _delete(r['id'] as int),
                              tooltip: '              ',
                            ),
                          );
                        },
                      ),
                    ),
                    Padding(
                      padding: const EdgeInsets.all(12.0),
                      child: OutlinedButton.icon(
                        onPressed: () => Navigator.of(context).pop(),
                        icon: const Icon(Icons.arrow_back),
                        label: const Text('Back'),
                      ),
                    ),
                  ],
                ),
    );
  }
}
```

В результате работы программы получился следующий вывод:

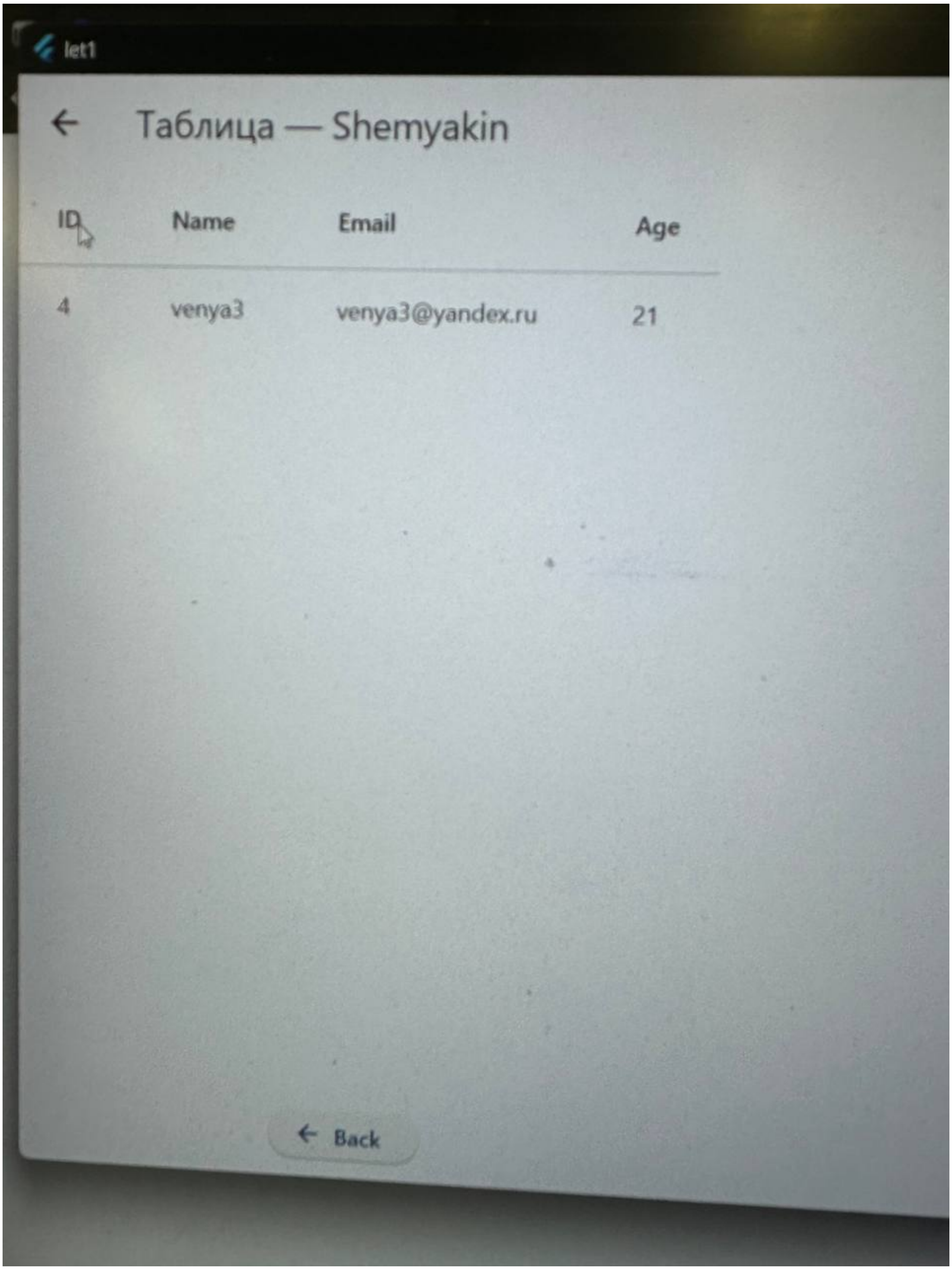
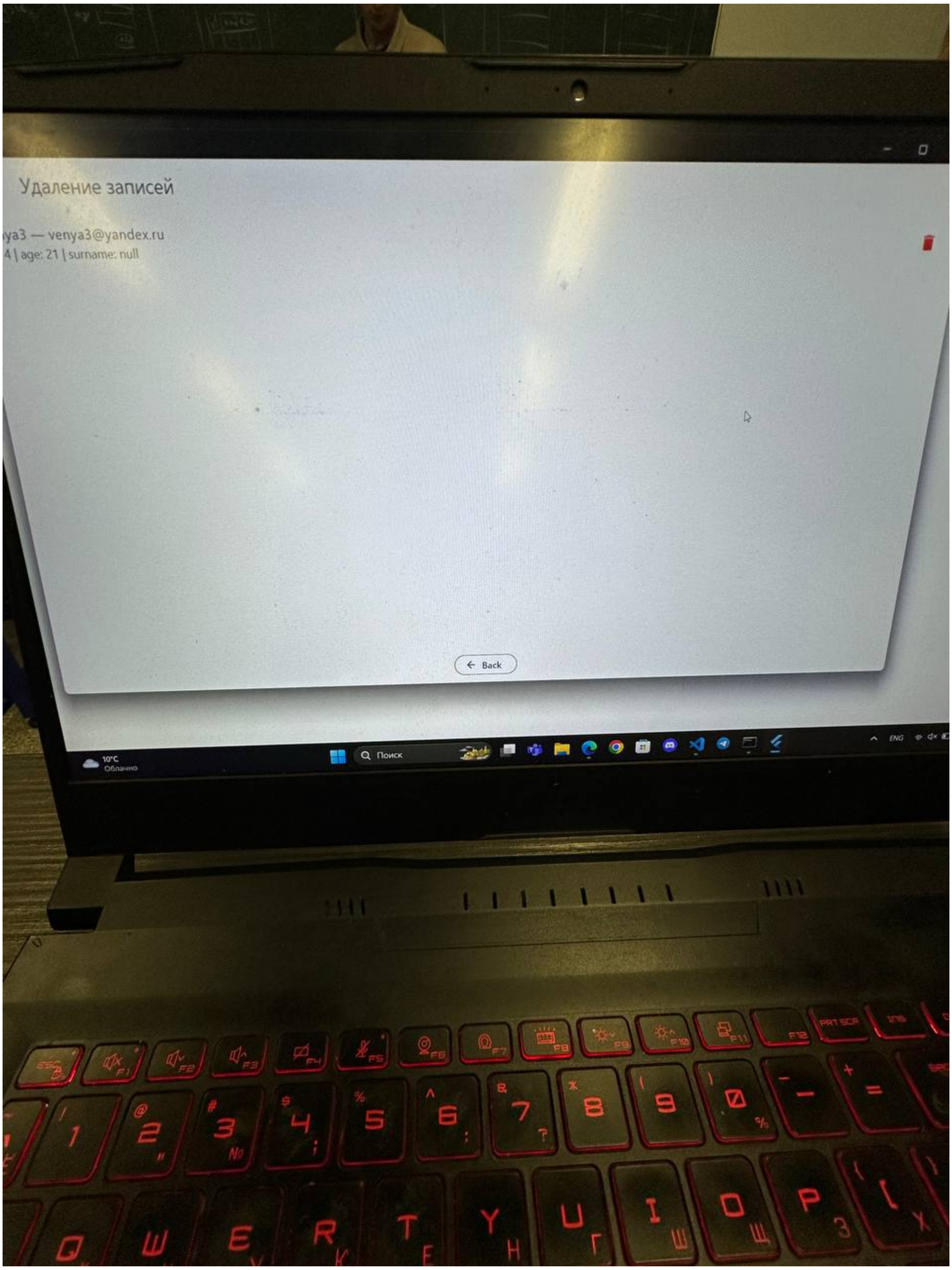


Таблица — Shemyakin

ID	Name	Email	Age
4	venya3	venya3@yandex.ru	21



Удаление записей

ya3 — venya3@yandex.ru
4 | age: 21 | surname: null

3 Заключение

В ходе лабораторной работы удалось реализовать подключение к базе данных MySQL и изменении данных в собственной таблице