
UNITAU
Universidade de Taubaté

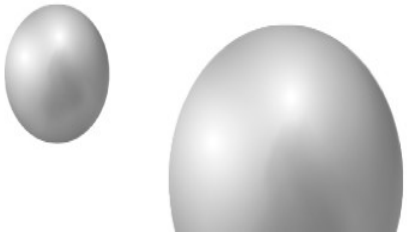

Persistência

Persistência

“Em ciência da computação, persistência se refere à característica de um estado que sobrevive ao processo que o criou. Sem essa capacidade, o estado só existiria na RAM, e seria perdido quando a RAM parasse (desligando-se o computador por exemplo).

Isso é conseguido na prática armazenando o estado como dados em armazenamento não volátil (como em um disco rígido ou memória flash), ou seja, via serialização dos dados para um formato armazenável e depois os salvando em um arquivo.”

https://pt.wikipedia.org/wiki/Persistência_(ciência_da_computação)



Persistência

Para as aplicações, persistência é armazenar os dados da aplicação em um arquivo local ou em um servidor.

O Flutter tem pacotes que permitem o armazenamento dos dados em diversas fontes, seja arquivo local gerenciado pela própria aplicação, em um banco de dados SQLite armazenado no próprio dispositivo, em um servidor de banco de dados externo ou em servidores na web.

DAO

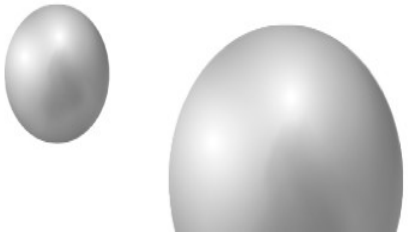
Objeto de acesso a dados (acrônimo do inglês **Data Access Object - DAO**), é um padrão para aplicações que utilizam persistência de dados implementado com linguagens de programação orientadas a objetos e arquitetura **MVC**. Para separar as regras de negócio das regras de acesso a banco de dados, todas as funcionalidades de bancos de dados, tais como obter conexões, mapear objetos para tipos de dados **SQL** ou executar comandos **SQL**, devem ser feitas por classes **DAO**.

A separação rigorosa entre a lógica de negócios e a lógica de persistência, permite que as alterações em qualquer uma das partes não implique em alterações na outra, desde que a interface entre elas não seja modificada.

DAO

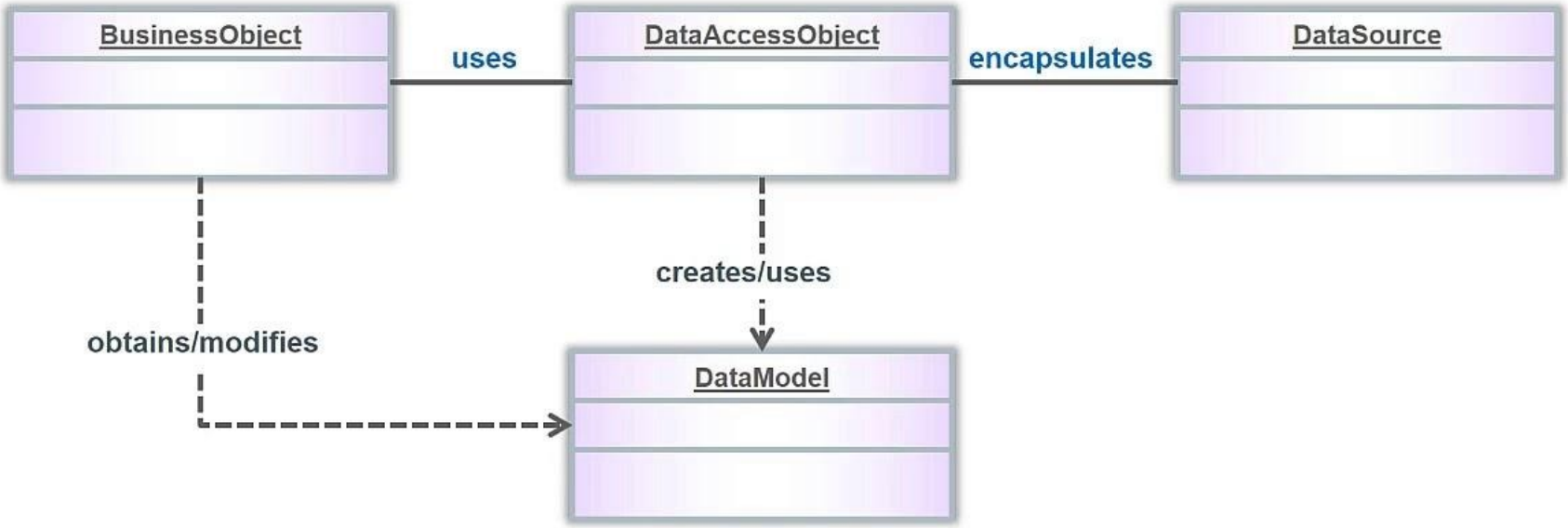
- Pode ser usada em uma vasta porcentagem de aplicações;
- Esconde todos os detalhes relativos a armazenamento de dados do resto da aplicação;
- Atua como um intermediário entre a aplicação e o banco de dados;
- Mitiga ou resolve problemas de comunicação entre a base de dados e a aplicação, evitando estados inconsistentes de dados.

https://pt.wikipedia.org/wiki/Objeto_de_acesso_a_dados



DAO

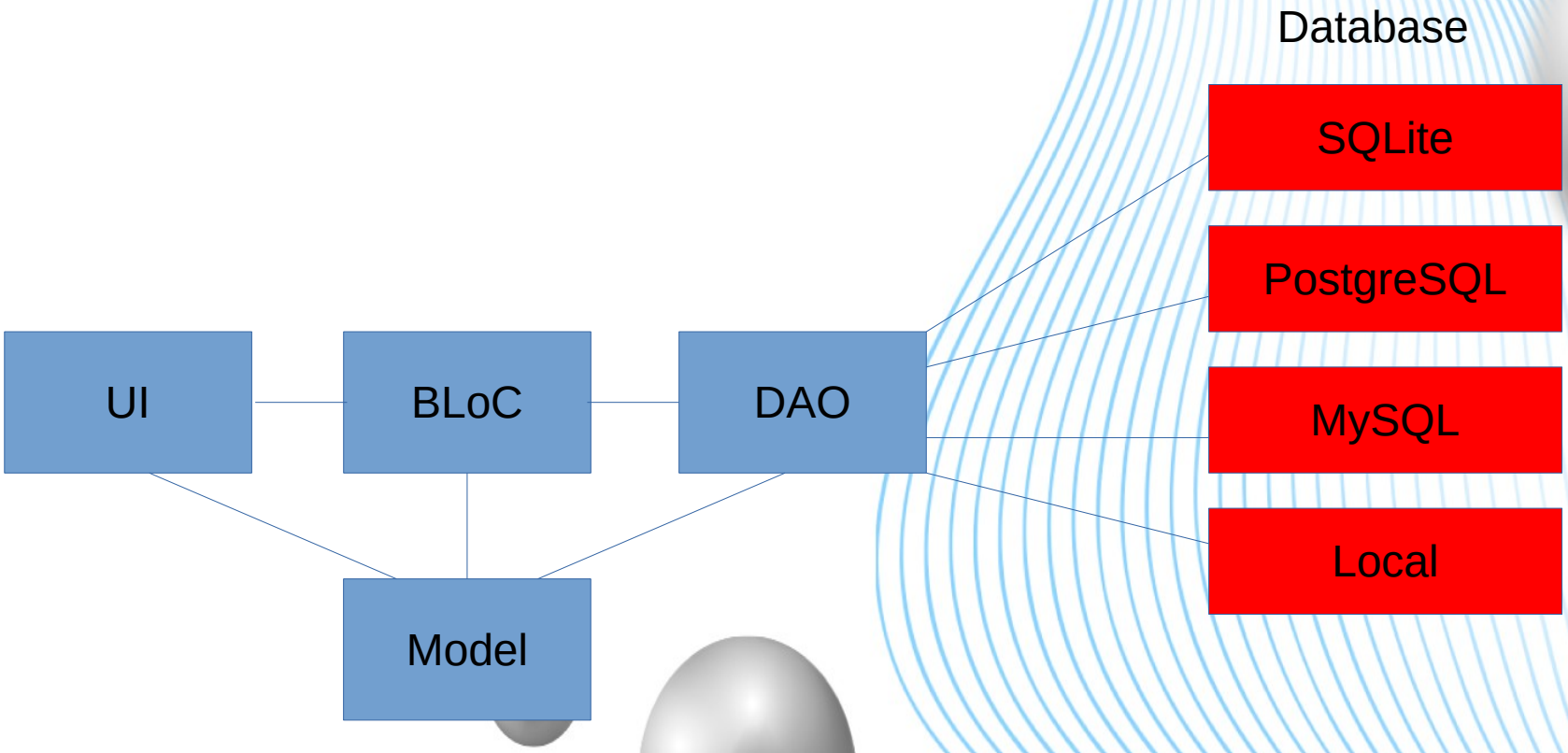
The Data Access Object (DAO) Pattern



https://www.youtube.com/watch?v=bigYRPn8VMg

DAO

Com essa divisão, será possível adaptar a aplicação a diversas camadas de persistência alterando apenas o classe de acesso ao banco de dados ou, quando necessário, com pequenas adaptações no DAO.

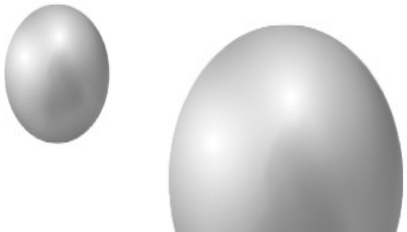


SQLite

“SQLite é uma biblioteca em linguagem C que implementa um banco de dados SQL embutido. Programas que usam a biblioteca SQLite podem ter acesso a banco de dados SQL sem executar um processo SGBD separado.

SQLite não é uma biblioteca cliente usada para conectar com um grande servidor de banco de dados, mas sim o próprio servidor. A biblioteca SQLite lê e escreve diretamente no arquivo de banco de dados no disco.”

https://pt.wikipedia.org/wiki/SQLite



SQLite

SQLite permite o armazenamento local de pequenas bases de dados. Não há um processo gerenciador do banco de dados. A própria aplicação lê e grava os dados localmente usando a biblioteca SQLite.

Embora erroneamente descrito como um SGBD em diversos textos, SQLite é na verdade um motor (engine) para gerenciamento de dados que fica embutido na aplicação. SQLite não tem diversos recursos disponíveis em SGBDs como gerenciamento de usuários e controle de acesso.

É indicado para pequenas aplicações que manipulam bases de dados leves armazenadas localmente.

Os sistemas operacionais Android e iOS implementam a biblioteca do SQLite, possibilitando que as aplicações para dispositivos móveis utilizem SQLite para armazenar dados localmente.

SQLite

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

Altere o arquivo **pubspec.yaml** para incluir as dependências de **sqflite** e **path_provider**:

```
dependencies:
  sqflite:
  path_provider: ^1.6.0
  flutter:
    sdk: flutter
```

Instale os pacotes adicionados nas dependências.

SQLite

Altere o arquivo main.dart:

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

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

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

SQLite

Crie o subdiretório lib/models e crie o arquivo student.dart no diretório lib/models:

```
//
// Modelo para dados do estudante
//
class Student {
  int id;
  String name;
  int age;
  String email;

  // Construtor padrao
  Student({this.id, this.name, this.age, this.email});

  // Construtor de um estudante vazio
  Student.empty() {
    id = 0;
    name = "";
    age = 0;
    email = "";
  }
```

SQLite

```
  // Construtor baseando em JSON
  factory Student.fromJson(Map<String, dynamic> json) {
    return Student(
      id: json['id'],
      name: json['name'],
      age: json['age'],
      email: json['email']
    );
  }

  // Converte um estudante para JSON
  Map<String, dynamic> toJson() {
    var map = {
      'id': id,
      'name': name,
      'age': age,
      'email': email
    };
    return map;
  }
}
```

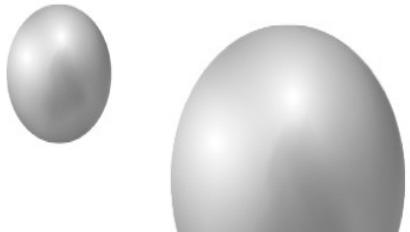
SQLite

Crie o subdiretório **lib/blocs** e crie o arquivo **student_bloc.dart** com as regras de negócio no diretório **lib/blocs**:

```
import 'dart:async';
import '../models/student.dart';
import '../dao/student_dao.dart';

class StudentBLoC {
  static final StudentBLoC _singleton =
StudentBLoC._internal();
  final _controller =
StreamController<List<Student>>.broadcast();
  final StudentDao _dao = StudentDao();

  factory StudentBLoC() => _singleton;

  StudentBLoC._internal();

  Stream<List<Student>> get stream => _controller.stream;
```

SQLite

```
  void dispose(){
    _controller.close();
    _dao.close();
  }
}
```



SQLite

Inclua o método para inicializar o acesso ao banco de dados e fazer a primeira carga dos registros dos alunos na classe StudentBloc:

```
void firstCall() async {
  bool _init = await _dao.init();
  if ( _init )
    getAll();
  else
    _controller.addError(_dao.errorMsg);
}
```

Inclua o método para carregar todos os registros dos alunos na classe StudentBloc:

```
void getAll() async {
  print("BLoC getAll");
  List<Student> _list = await _dao.getAll();
  if (_list != null)
    _controller.add(_list);
  else
    _controller.addError(_dao.errorMsg); }
```

SQLite

Inclua o método para carregar o registro do aluno de um determinado id na classe StudentBloc:

```
void getByID(int _id) async {
  print("BLoC getByID");
  List<Student> _list = List<Student>();
  Student _student = await _dao.getByID(_id);
  if (_student != null) {
    _list.add(_student);
    _controller.add(_list);
  } else
    _controller.addError(_dao.errorMsg);
}
```

SQLite

Inclua o método para inserir um aluno na classe StudentBloc:

```
Future<int> insert(Student student) async {
  print("BLoC insert");
  int ret = await _dao.insert(student);
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_dao.errorMsg);
}
```

Inclua o método para alterar um aluno na classe StudentBloc:

```
Future<int> update(Student student) async {
  print("BLoC update");
  int ret = await _dao.update(student);
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_dao.errorMsg); }
```

SQLite

Inclua o método para deletar um aluno na classe StudentBloc:

```
Future <int> delete(int _id) async {
  print("BLoC Delete");
  int ret = await _dao.delete(_id);
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_dao.errorMsg);
}
```

SQLite

Crie o subdiretório **lib/dao** e crie o arquivo **student_dao.dart** com a classe para acesso aos dados no diretório **lib/dao**:

```
import 'dart:async';
import '../models/student.dart';
import '../database/database.dart';

class StudentDao {
  final DatabaseHelper _dbHelper = DatabaseHelper();
  String _errorMsg = "";

  String get errorMsg => _errorMsg;

  close() {
    _dbHelper.close();
  }
}
```

SQLite

Inclua o método para inicializar o acesso ao banco de dados na classe StudentDao:

```
  Future<bool> init() async {
    _errorMsg = "";
    bool ret = await
DatabaseHelper.initDb().catchError((_error) {
      _errorMsg = _error;
    });
    ret ??= false;
    return ret;
  }
```

SQLite

Inclua o método para carregar todos os registros dos alunos na classe StudentDao:

```
Future<List<Student>> getAll() async {
  print("DAO getAll");
  _errorMsg = "";
  List<Map> _maps = await DatabaseHelper.getAll(
    'student'
  ).catchError((_error) {
      _errorMsg = _error;
  });
  if (_maps != null) {
    List<Student> _students = [];
    if (_maps.length > 0) {
      for (int i = 0; i < _maps.length; i++) {
        _students.add(Student.fromJson(_maps[i]));
      }
    }
    return _students;
  } else
    return null;
}
```

SQLite

Inclua o método para carregar o registro do aluno de um determinado id na classe StudentDao:

```
Future<Student> getByID(int _id) async {
  print("DAO getByID");
  Student ret;
  _errorMsg = "";
  var response = await DatabaseHelper.getByID(
    "student",
    _id
  ).catchError((_error) {
    _errorMsg = _error;
  });
  if ( _errorMsg == "" ) {
    if (response.isNotEmpty)
      ret = Student.fromJson(response);
    else
      ret = Student.empty();
  }
  return ret;
}
```

SQLite

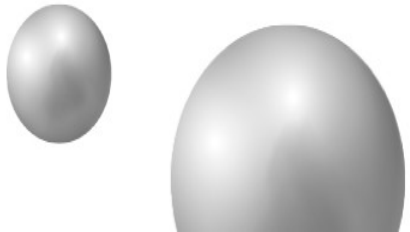
Inclua o método para inserir um aluno na classe StudentDao:

```
Future<int> insert(Student _student) async {
  print("DAO insert");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.insert(
    'student',
    _student.toJson()
  ).catchError((_error) {
    _errorMsg = _error;
  });
  return ret;
}
```

SQLite

Inclua o método para alterar um aluno na classe StudentDao:

```
Future<int> update(Student _student) async {
  print("DAO update");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.update(
    'student',
    _student.toJson(),
  ).catchError((_error) {
    _errorMsg = _error;
  });
  return ret;
}
```



SQLite

Inclua o método para deletar um aluno na classe StudentDao:

```
Future<int> delete(int _id) async {
  print("DAO delete");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.delete(
    'student',
    _id,
  ).catchError((_error) {
    _errorMsg = _error;
  });
  return ret;
}
```

SQLite

Crie o subdiretório **lib/database** e crie o arquivo **database.dart** com a classe para acesso ao SQLite no diretório **lib/database**:

```
import 'package:sqflite/sqflite.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static Database _db;

  void close() {
    print("CloseDB");
    if (_db != null) _db.close();
  }
}
```

SQLite

Inclua o método para inicializar o acesso ao banco de dados na classe DatabaseHelper:

```
static Future<bool> initDb() async {
  if ( _db == null ) {
    print("InitDB");
    String _errorMsg = "";
    String _path = await getDatabasesPath() + 'example';
    _db = await openDatabase(
      _path,
      version: 1,
      onCreate: _onCreate
    ).catchError((_error) {
      _errorMsg = _error.message;
    });
    if ( _db == null )
      return Future<bool>.error(_errorMsg);
  }
  return true;
}
```

SQLite

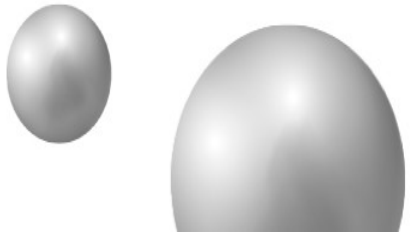
Inclua o método para criar a tabela no banco de dados na classe DatabaseHelper:

```
static void _onCreate(Database db, int version) async {
  print("CreateDB");
  await db.execute('''CREATE TABLE student(
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER,
    email TEXT)'''
  );
}
```

SQLite

Inclua o método para carregar todos os registros de uma tabela na classe DatabaseHelper:

```
static Future<List<Map>> getAll(String _table) async {
  print("Database getAll");
  List<Map> ret;
  String _errorMsg = "";
  ret = await _db.query(_table).catchError((_error) {
    _errorMsg = _error.message;
  });
  if (ret != null)
    return ret;
  else
    return Future<List<Map>>.error(_errorMsg);
}
```



SQLite

Inclua o método para carregar o registro com um determinado id de uma tabela na classe DatabaseHelper:

```
static Future<Map> getByID(String _table, int _id) async {
  print("Database getByID");
  Map ret;
  String _errorMsg = "";
  var _rows = await _db.query(
    _table,
    where: "id = ?",
    whereArgs: [_id]
  ).catchError((_error) {
    _errorMsg = _error.message;
  });
  if ( _rows != null ) {
    if (_rows.isNotEmpty)
      ret = _rows.first;
    else ret = Map();
    return ret;
  } else
    return Future<Map>.error(_errorMsg);
}
```

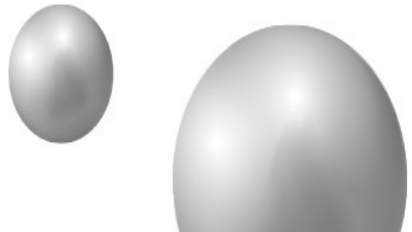
SQLite

Inclua o método para inserir um registro em uma tabela na classe DatabaseHelper:

```
static Future<int> insert(String _table, Map _map) async {
  print("Database insert");
  int ret;
  String _errorMsg = "";
  var _max = await _db.rawQuery(
    "SELECT MAX(id)+1 as id FROM ${_table}"
  ).catchError((_error) {
    _errorMsg = _error.message;
  });
  if (_errorMsg == "") {
    int _id = _max.first["id"];
    _map['id'] = _id;
    ret = await _db.insert(
      _table,
      _map
    ).catchError((_error) {
      _errorMsg = _error.message;
    });
  }
```

SQLite

```
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_errorMsg);
}
```



SQLite

Inclua o método para alterar um registro em uma tabela na classe DatabaseHelper:

```
static Future<int> update(String _table, Map _map) async {
  print("Database update");
  int ret;
  String _errorMsg = "";
  ret = await _db.update(
    _table,
    _map,
    where: 'id = ?',
    whereArgs: [_map['id']],
  ).catchError((_error) {
    _errorMsg = _error.message;
  });
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_errorMsg);
}
```

SQLite

Inclua o método para deletar um registro em uma tabela na classe DatabaseHelper:

```
static Future<int> delete(String _table, int _id) async {
  print("Database delete");
  int ret;
  String _errorMsg = "";
  ret = await _db.delete(
    _table,
    where: 'id = ?',
    whereArgs: [_id]
  ).catchError((_error) {
    _errorMsg = _error.message;
  });
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_errorMsg);
}
```

SQLite

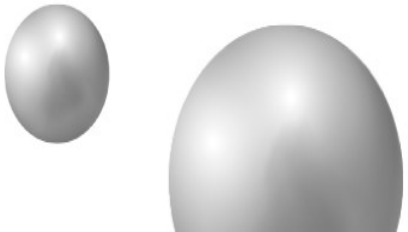
Crie o subdiretório **lib/utils** e crie o arquivo **wait.dart** no diretório **lib/utils**:

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

//
// Apresenta um indicador de espera por uma operacao
//
class WaitWidget extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Container(
        child: Center(
          child: CircularProgressIndicator(),
        ),
        color: Colors.white.withOpacity(0.8));
  }
}
```

SQLite

Crie o arquivo confirm.dart no diretório lib/utils:

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

Future<bool> confirm(BuildContext context, String _msg)
async {
  return await showDialog(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) {
      return AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20.0),
        ),
```



SQLite

```
        title: Row(
          children: [
            Icon(Icons.help,color: Colors.blue,size: 48.0,),
            Expanded(
              child: Text(
                'Confirmacao',
                style: TextStyle(fontWeight:
FontWeight.bold),
                textAlign: TextAlign.center,
              ),
            ),
          ]
        ),
        content: Text(_msg),
```

SQLite

```
        actions: <Widget>[
          FlatButton(
            onPressed: () =>
Navigator.of(context).pop(true),
            child: const Text("OK")
          ),
          FlatButton(
            onPressed: () =>
Navigator.of(context).pop(false),
            child: const Text("CANCEL"),
          ),
        ],
      );
    },
  );
}
```

SQLite

Crie o arquivo showerror.dart no diretório lib/utils:

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

void showError(BuildContext context, String _msg) async {
  return await showDialog<void>(
    context: context,
    barrierDismissible: false,
    builder: (BuildContext context) {
      return AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20.0),
        ),
```

SQLite

```
        title: Row(
            children: [
              Icon(Icons.error,color: Colors.red,size:
48.0,),
              Expanded(
                child: Text(
                  'Erro',
                  style: TextStyle(fontWeight:
FontWeight.bold),
                  textAlign: TextAlign.center,
                ),
              ),
            ]
        ),
        content: Text(_msg),
```

SQLite

```
        actions: <Widget>[
          FlatButton(
            child: Text('Ok'),
            onPressed: () => Navigator.of(context).pop(),
          ),
        ],
      );
    },
  );
}
```

SQLite

Crie o subdiretório **lib/pages** e crie o arquivo **home.dart** no diretório **lib/pages**:

```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'detail.dart';
import '../models/student.dart';
import '../blocs/student_bloc.dart';
import '../utils/confirm.dart';
import '../utils/showerror.dart';
import '../utils/wait.dart';

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

SQLite

Crie a classe para gerenciar o estado da tela da aplicação no arquivo home.dart:

```
class _HomeState extends State<Home> {
  // Chave para o Scaffold
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
  // Widget de indicacao de espera
  final WaitWidget _waitWidget = WaitWidget();
  // BLoC para estudantes
  StudentBLoC _bloc = StudentBLoC();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      key: _scaffoldKey,
      appBar: AppBar(
        title: Text("SQLite"),
```

SQLite

```
    actions: <Widget>[
      IconButton(
        icon: Icon(Icons.add),
        tooltip: 'Inserir',
        onPressed: () => _detail(context, null),
      ),
    ],
  ),
  body: StreamBuilder<List<Student>>(
    stream: _bloc.stream,
    builder: (context, snapshot) {
      print("Recriando lista de alunos");
      if (snapshot.hasData ) {
        if (snapshot.data.length>0) {
          return RefreshIndicator(
            onRefresh: _refreshData,
            child: ListView.builder(
              itemCount: snapshot.data.length,
```

SQLite

```
                    itemBuilder: (BuildContext ctxt, int
index) {
                      return Dismissible(
                        key: Key(
                          "${snapshot.data[index].id}$
{snapshot.data[index].name}"
                        ),
                        onDismissed: (direction) {

_scaffoldKey.currentState.showSnackBar(
                            SnackBar(content:
                              Text("Removido $
{snapshot.data[index].name}")
                            )
                          );
                          setState(() =>
snapshot.data.removeAt(index));
                        },
```

SQLite

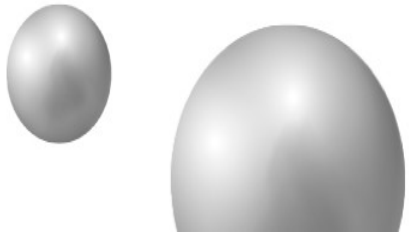
```
                        background: Container(color:
Colors.red),
                        child: ListTile(
                          title: Text('$
{snapshot.data[index].name}'),
                          subtitle: Text('Age: $
{snapshot.data[index].age}'),
                          trailing: Text('ID: $
{snapshot.data[index].id}'),
                          onTap: () =>
                            _detail(context,
snapshot.data[index].id),
                        ),
                        confirmDismiss: (DismissDirection
direction) async {
                          bool delete = await confirm(context,
                            "Deletar $
{snapshot.data[index].id} ${snapshot.data[index].name}?"
                          );
```

SQLite

```
            if (delete) {
              String _errorMsg;
              int _deleted = await _bloc.delete(
                snapshot.data[index].id
              ).catchError((_error) {
                _errorMsg = _error;
              });
              if ( _deleted == null ) {
                delete = false;
                showError(context, _errorMsg);
              } else {
                if (_deleted != 1) {
                  delete = false;
                  print("Erro na delecao"); }
              }
            }
            return delete;
          },
        );
      }
    ),
);
```

SQLite

```
            } else {
              return Center(
                child: Text("No Data",
                  style:
Theme.of(context).textTheme.display1,
                )
              );
            }
          } else if (snapshot.hasError) {
            return Center(
              child: Text('Error: ${snapshot.error}',
                style: Theme.of(context).textTheme.display1,
              ),
            );
          } else {
            _bloc.firstCall();
            return _waitWidget;
          }
        }
      ),
    );
  }
```



SQLite

```
  Future _refreshData() async {
    _bloc.getAll();
  }

  _detail(BuildContext context, int id) async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => Detail(id:
id)),
    );
    _bloc.getAll();
  }

  dispose() {
    _bloc.dispose();
  }
}
```

SQLite

Crie o arquivo detail.dart no diretório lib/pages:

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'dart:convert';
import '../models/student.dart';
import '../blocs/student_bloc.dart';
import '../utils/showerror.dart';
import '../utils/wait.dart';

//
// Classe para edicao do estudante
//
class Detail extends StatelessWidget {
  Detail({Key key, @required this.id}) : super(key: key);
```

SQLite

```
  // ID do usuario a editar
  final int id;
  // Chave para o formulario
  final GlobalKey<FormState> _formStateKey =
GlobalKey<FormState>();
  // Chave para o Scaffold do formulario
  final GlobalKey<ScaffoldState> _scaffoldKey =
GlobalKey<ScaffoldState>();
  // Widget de indicacao de espera
  final WaitWidget _waitWidget = WaitWidget();
    // BLoC para estudantes
  StudentBLoC _bloc = StudentBLoC();
  // Dados do estudante
  Student _student;
```

SQLite

```
@override
Widget build(BuildContext context) {
  String title;
  if (id ==  null) {
    title = "Incluir";
    _student = Student.empty();
  } else {
    _student = null;
    title = "Editar";
  }
  return Scaffold(
    key: _scaffoldKey,
    appBar: AppBar(
      title: Text(title),
    ),
```

SQLite

```
body: Builder(builder: (BuildContext context) {
  if (id == null) {
    return _editStudent();
  } else {
    return StreamBuilder<List<Student>>(
      stream: _bloc.stream,
      builder: (context, snapshot) {
        print("Pesquisando aluno");
        if (snapshot.hasData ) {
          if (snapshot.data.length > 0) {
            _student = snapshot.data.first;
            return _editStudent();
          } else {
            return Text('Erro carregando estudante');
          }
```

SQLite

```
            } else if (snapshot.hasError) {
              return Center(
                child: Text(
                  'Error: ${snapshot.error}',
                  style:
Theme.of(context).textTheme.display1,
                ),
              );
            } else {
              _bloc.getByID(id);
              return _waitWidget;
            }
          }
        );
      }
    }),
```

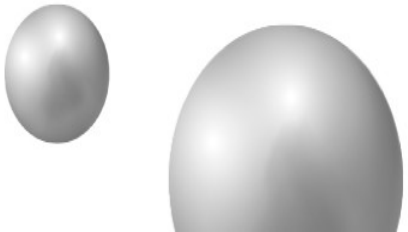
SQLite

```
    bottomNavigationBar: ButtonBar(
      alignment: MainAxisAlignment.spaceEvenly,
      children: <Widget>[
        RaisedButton(
            onPressed: () => _reset(),
            child: Text('Reset')
        ),
        RaisedButton(
            onPressed: () => _submit(context),
            child: Text('Save')
        ),
        RaisedButton(
          onPressed: () => Navigator.pop(context, null),
          child: Text('Cancel')
        ),
      ],
    ),
  );
}
```

SQLite

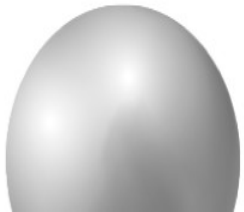
Inclua o método para construir os widgets para editar os dados do estudante na classe Detail:

```
//
// Edita o estudante
//
SafeArea _editStudent() {
  SafeArea ret = SafeArea(
    child: Column(
      children: <Widget>[
        Form(
          key: _formStateKey,
          autovalidate: true,
          child: Padding(
            padding: EdgeInsets.all(16.0),
            child: Column(
              children: <Widget>[
```



SQLite

```
                    TextFormField(
                      initialValue: _student.name,
                      decoration: InputDecoration(labelText:
'Nome',),
                      validator: (value) =>
_validateName(value),
                      onSaved: (value) => _student.name =
value,
                    ),
                    TextFormField(
                      initialValue: _student.age.toString(),
                      inputFormatters:

[WhitelistingTextInputFormatter.digitsOnly],
                      keyboardType: TextInputType.number,
                      decoration: InputDecoration(labelText:
'Idade'),
                      onSaved: (value) => _student.age =
int.parse(value),
                    ),
```



SQLite

```
                    TextFormField(
                      initialValue: _student.email,
                      decoration: InputDecoration(labelText:
'Email',),
                      onSaved: (value) => _student.email =
value,
                    ),
                  ],
                ),
              ),
            ),
          ],
        ),
      );
      return ret;
    }
```

SQLite

Inclua o método para validar o nome na classe Detail:

```
//
// Valida o nome
//
String _validateName(String value) {
  String ret = null;
  value = value.trim();
  if (value.isEmpty)
    ret = "Nome e obrigatorio";
  return ret;
}
```

Inclua o método para resetar os dados do formulário na classe Detail:

```
//
// Reseta os dados do formulario
//
void _reset() {
  if (_student != null)
    _formStateKey.currentState.reset();
}
```

SQLite

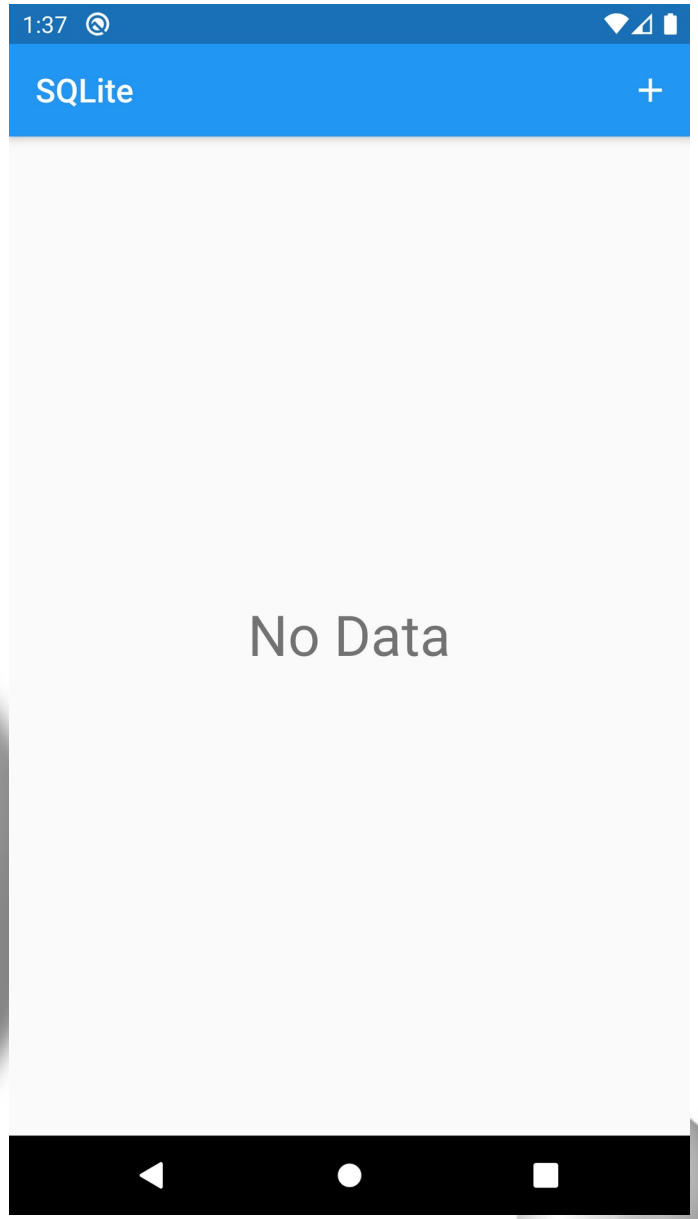
Inclua o método para salvar os dados na classe Detail:

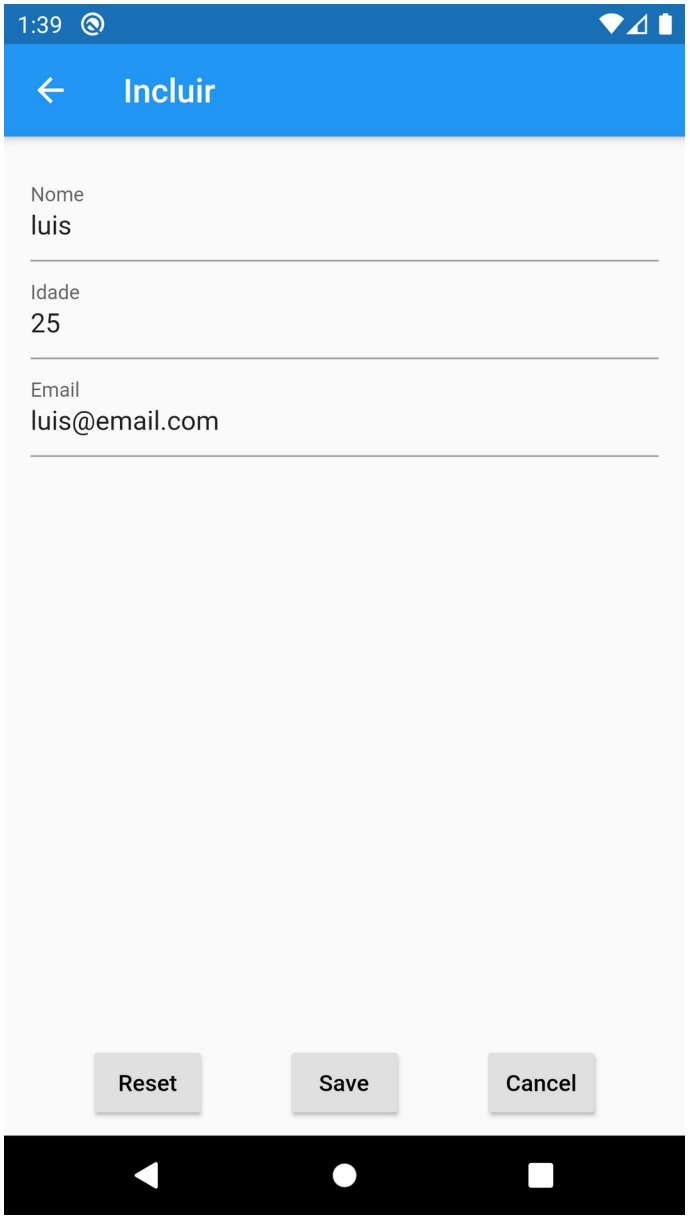
```
//
// Verifica e salva os dados do formulario
//
void _submit(BuildContext context) async {
  if ( _student != null ) {
    print("Salvando");
    String _errorMsg;
    if (_formStateKey.currentState.validate()) {
      _formStateKey.currentState.save();
      var value;
      if (id == null)
        value = await _bloc.insert(_student)
          .catchError((_error) {
            _errorMsg = _error;
          });
      else
        value = await _bloc.update(_student)
          .catchError((_error) {
            _errorMsg = _error;
          });
```

SQLite

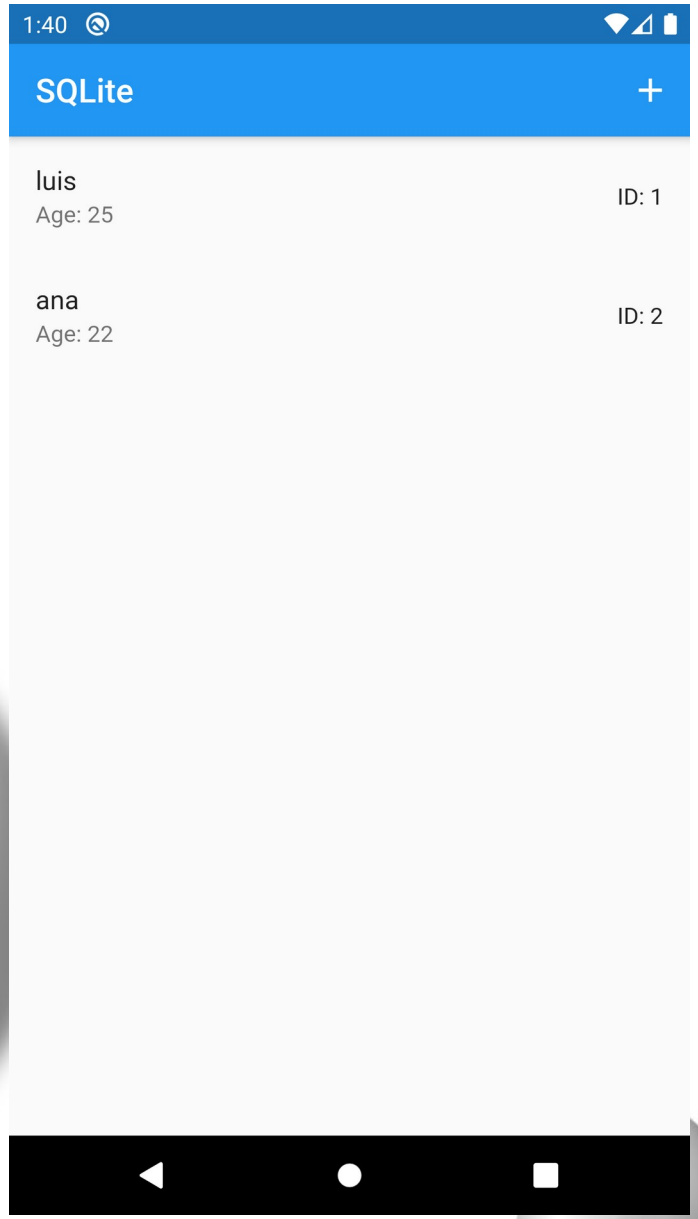
```
        value ??= 0;
        if (value > 0) {
          _scaffoldKey.currentState.showSnackBar(
              SnackBar(content: Text('Salvo ${_student.name}'))
          );
          Navigator.pop(context, true);
        } else
          showError(context, _errorMsg);
      }
    }
  }
}
```

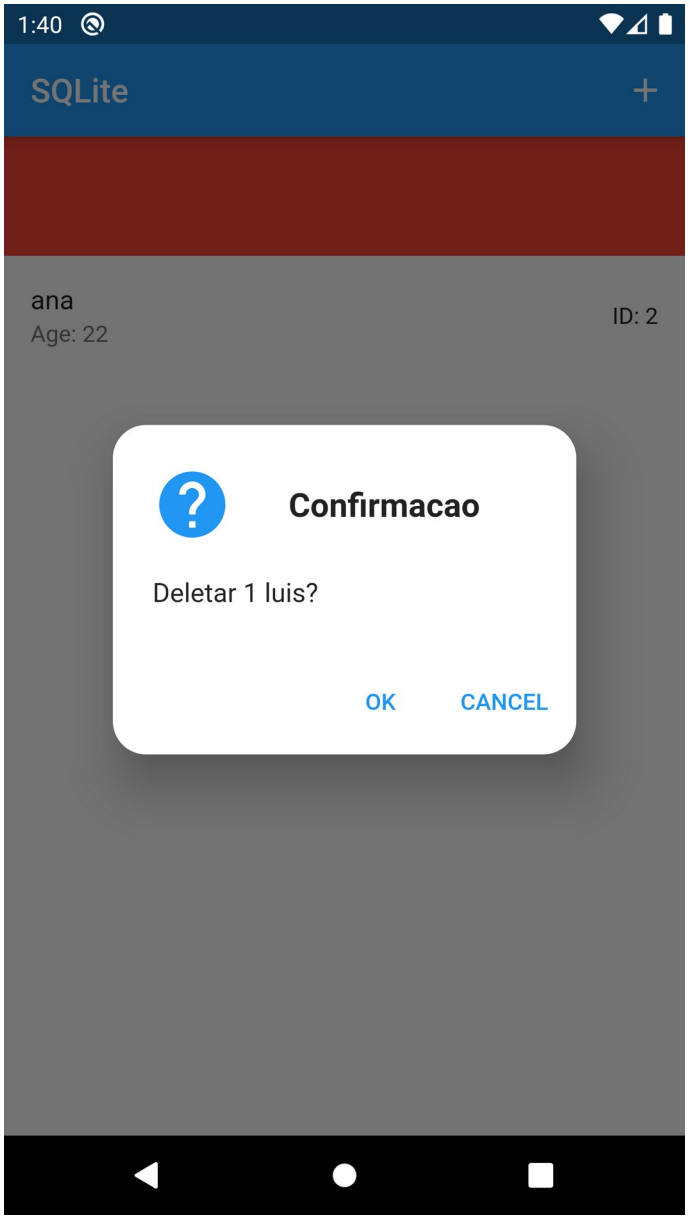
SQLite


1:37
SQLite
No Data


1:39
Incluir
Nome
luis
Idade
25
Email
luis@email.com
Reset
Save
Cancel

SQLite


1:40
SQLite
luis
Age: 25
ID: 1
ana
Age: 22
ID: 2

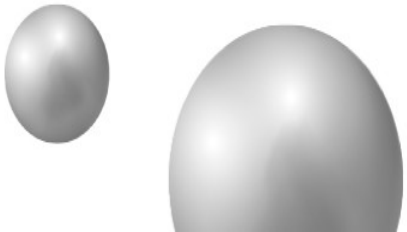

1:40
SQLite
ana
Age: 22
ID: 2
Confirmacao
Deletar 1 luis?
OK
CANCEL

SGBDs

É possível utilizar um pacote para acesso à banco de dados para conectar diretamente um SGBD e executar os comandos para acessar os dados do servidor.

Porém essa abordagem não é a mais adequada em aplicações executadas pela Internet pois deixar o SGBD diretamente exposto na Internet não é uma boa prática de segurança. Nesses casos, o ideal é que o SGBD não seja acessível pela Internet e a aplicação acesse os dados através de algum middleware.

Em aplicações rodando dentro da rede da empresa é aceitável que a aplicação acesse diretamente o SGBD.

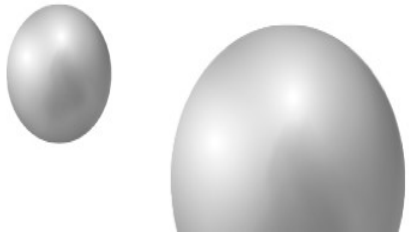


SGBDs

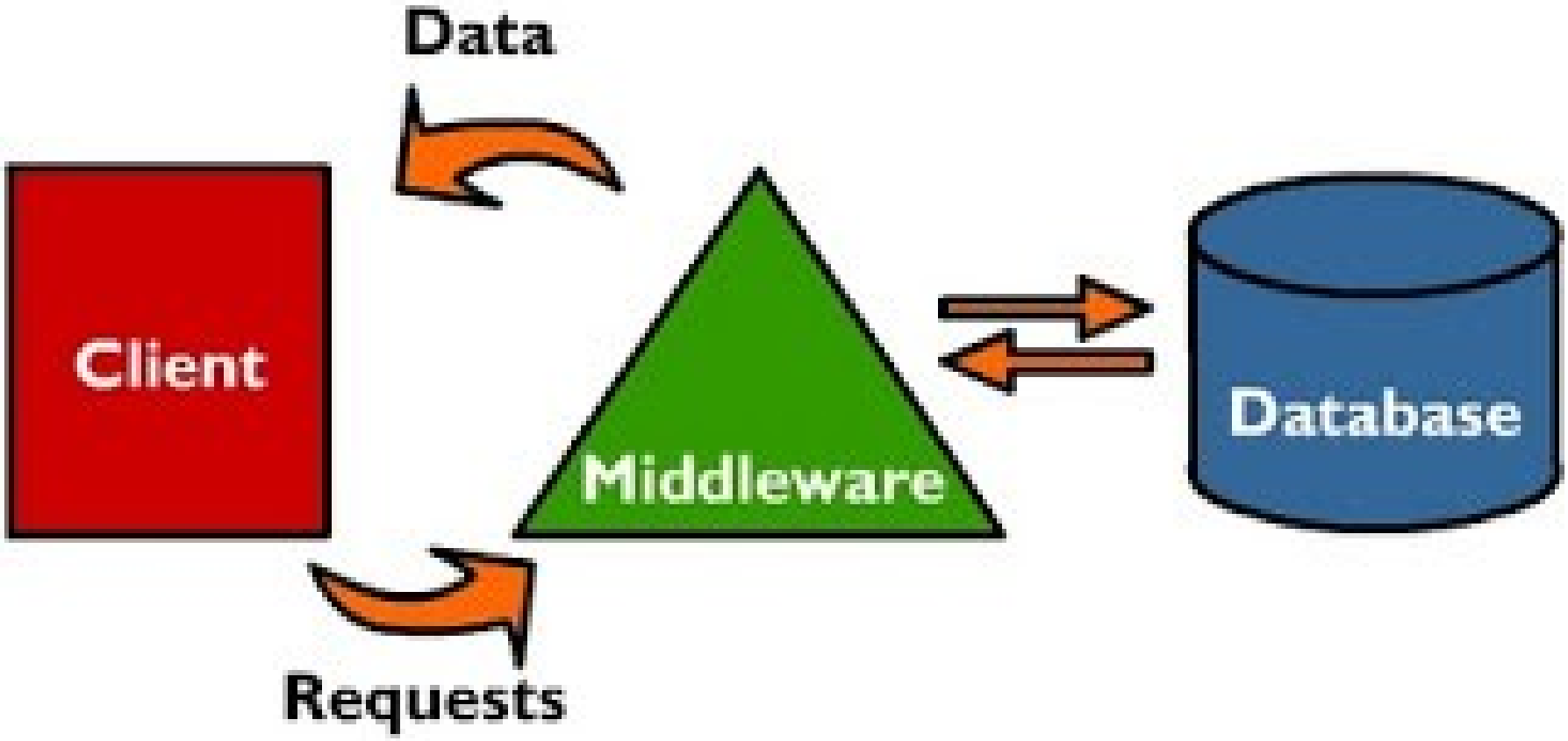
“Middleware é um software que fornece serviços e recursos comuns a aplicações. Gerenciamento de dados, serviços de aplicações, sistema de mensageria, autenticação e gerenciamento de APIs são recursos comumente operados por um software de middleware.

Com o middleware, os desenvolvedores podem criar aplicações com mais facilidade e eficiência, pois esse tipo de software tem o papel de conectar aplicações, dados e usuários.”

https://www.redhat.com/pt-br/topics/middleware/what-is-middleware



SGBDs

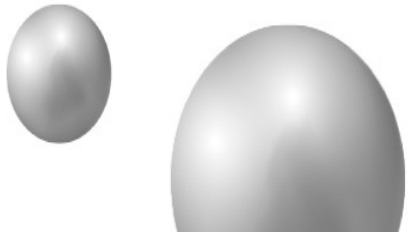


http://middlewaremania.blogspot.com/2012/11/what-is-middleware-all-about.html

PostgreSQL

Para a aplicação utilizando PostgreSQL será necessário criar um usuário e um banco de dados para a aplicação. No exemplo será utilizado o banco de dados mobile, usuário mobile e senha mobile:

```
> psql -U postgres
psql (12.2, server 11.7)
Type "help" for help.

postgres=# CREATE USER mobile WITH password 'mobile';
CREATE ROLE
postgres=# CREATE DATABASE mobile OWNER mobile;
CREATE DATABASE
postgres=#
```



PostgreSQL

É necessário garantir que as configurações de segurança do SGBD, no arquivo **pg_hba.conf** do diretório do banco de dados, permitam o acesso ao banco criado pela interface de loopback (127.0.0.1). Se for usado um dispositivo móvel para executar a aplicação, esse dispositivo também deve ter permissão de acesso ao banco de dados. Exemplo:

```
# TYPE   DATABASE    USER        ADDRESS                 METHOD
host     all         all         127.0.0.1/32            md5
host     all         all         192.168.0.0/24          md5
```

É necessário conectar no banco criado com o usuário que será utilizado na aplicação, e criar a tabela **student**:

```
CREATE TABLE student(
                    id      INTEGER PRIMARY KEY,
                    name    TEXT,
                    age     INTEGER,
                    email   TEXT
                    );
```

PostgreSQL

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

Altere o arquivo **pubspec.yaml** para incluir a dependência de **postgresql2**:

```
dependencies:
  postgresql2:
  flutter:
    sdk: flutter
```

Instale o pacote adicionado nas dependências.

PostgreSQL

Copie toda a estrutura de subdiretórios e arquivos do subdiretório **lib** da aplicação **persistencia_sqlite**.

Altere o nome da aplicação no arquivo **home.dart** no subdiretório **lib/pages**:

```
appBar: AppBar(
  title: Text("PostgreSQL"),
```

PostgreSQL

A uri de conexão com o banco de dados do pacote tem o formato:

```
postgres://<user>:<password>@<address>:<port>/<database>
```

O endereço 10.0.2.2 é o padrão do emulador do Android Studio para acesso a interface de loopback (127.0.0.1) do computador que está executando o emulador.

Altere o arquivo **database.dart** no subdiretório **lib/database**:

```
import 'package:postgresql2/postgresql.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static var uri =
'postgres://mobile:mobile@10.0.2.2:5432/mobile';
  static Connection _conn;
```

PostgreSQL

```
static Future<bool> initDb() async {
  if ( _conn == null ) {
    String _errorMsg;
    print("Conectando");
    _conn = await connect(uri).catchError((_error) {
      _errorMsg = _error.message;
    });
    if ( _conn == null )
      return Future<bool>.error(_errorMsg);
    else
      print("Conectado");
  }
  return true;
}
```

PostgreSQL

```
static Future<List<Map>> getAll(String _table) async {
  print("Database getAll");
  List<Map> ret = List<Map>();
  String _errorMsg = "";
  String _sql = "SELECT * FROM ${_table}";
  var _rows = await _conn.query(_sql)
    .toList().catchError((_error) {
      _errorMsg = _error.message+" on query "+_sql;
    });
  if (_errorMsg == "") {
    for (Row _row in _rows) {
      ret.add(_row.toMap());
    }
    return ret;
  } else
    return Future<List<Map>>.error(_errorMsg);
}
```

PostgreSQL

```
  static Future<Map> getByID(String _table, int _id) async {
    print("Database getByID");
    Map ret;
    String _errorMsg = "";
      String _sql = "SELECT * FROM ${_table} WHERE id='$
{_id}'";
    var _rows = await _conn.query(_sql)
      .toList().catchError((_error) {
        _errorMsg = _error.message+" on query "+_sql;
      });
    if ( _rows != null ) {
      if (_rows.isNotEmpty)
        ret = _rows.first.toMap();
      else
        ret = Map();
      return ret;
    } else
      return Future<Map>.error(_errorMsg);
  }
```

PostgreSQL

```
  static Future<int> insert(String _table, Map _map) async {
    print("Database insert");
    int ret;
    String _errorMsg = "";
    String _sql = "SELECT COALESCE(MAX(id),0)+1 as id FROM $
{_table}";
    var _rows = await _conn.query(
      _sql
    ).toList().catchError((_error) {
      _errorMsg = _error.message+" on query "+_sql;
    });
    if ( _rows != null ) {
      _map['id'] = _rows.first[0];
      String _fields = "";
      String _values = "";
      _map.forEach((_key, _value) {
        if (_fields != "")
          _fields += ",";
        _fields += _key;
        if (_values != "")
          _values += ",";
        _values += "'${_value}'"; });
```

PostgreSQL

```
      String _sql = "INSERT INTO ${_table} (${_fields})
VALUES (${_values})";
      ret = await _conn.execute(_sql).catchError((_error) {
        _errorMsg = _error.message+" on query "+_sql;
      });
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

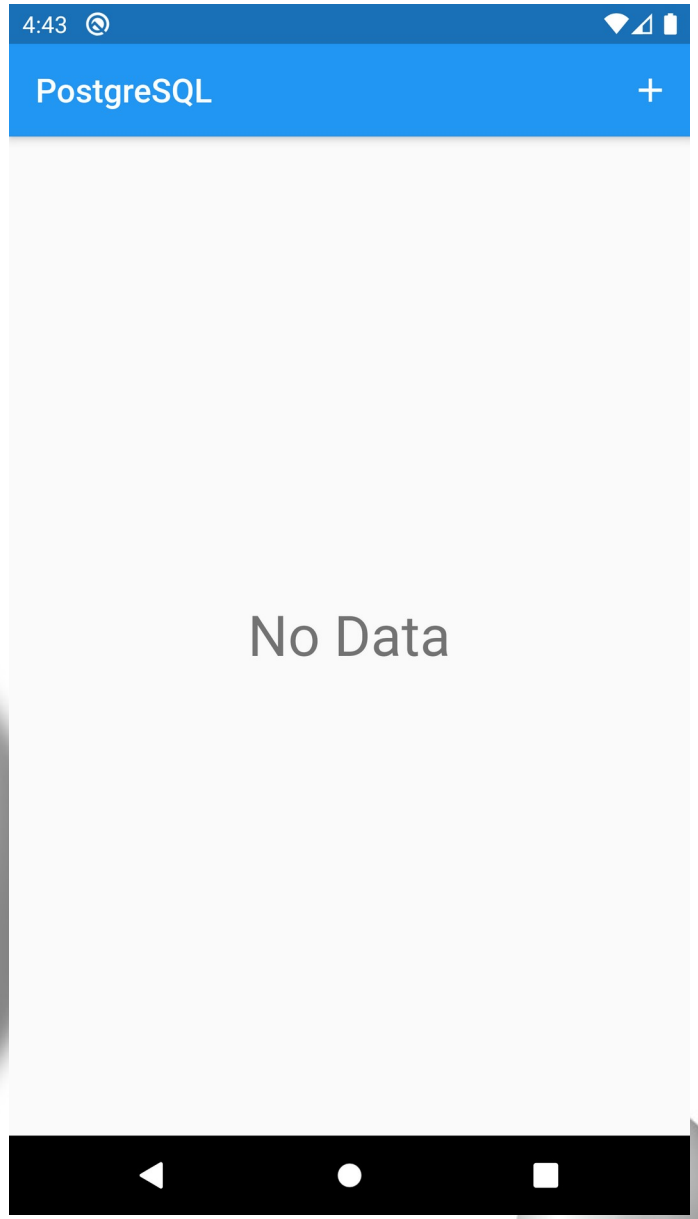
PostgreSQL

```
  static Future<int> update(String _table, Map _map) async {
    print("Database update");
    int ret;
    String _errorMsg = "";
    String _fields = "";
    _map.forEach((_key, _value) {
      if (_key != "id") {
        if (_fields != "")
          _fields += ",";
        _fields += "${_key}='${_value}'";
      }
    });
      String _sql = "UPDATE ${_table} SET ${_fields} WHERE
id='${_map['id']}'";
    ret = await _conn.execute(_sql).catchError((_error) {
      _errorMsg = _error.message+" on query "+_sql;
    });
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

PostgreSQL

```
  static Future<int> delete(String _table, int _id) async {
    print("Database delete");
    int ret;
    String _errorMsg = "";
       String _sql = "DELETE FROM "+_table+" WHERE id='${_id}'";
    ret = await _conn.execute(_sql).catchError((_error) {
      _errorMsg = _error.message+" on query "+_sql;
    });
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }

  void close() {
    print("CloseDB");
    _conn.close();
  }
}
```

PostgreSQL


4:43
PostgreSQL
+
No Data


4:45
PostgreSQL
+
ana
Age: 22
ID: 1
luis
Age: 21
ID: 2

PostgreSQL

Para utilizar a aplicação em um device e não no emulador, é necessário utilizar o endereço do servidor de banco de dados na uri de conexão. Usaremos um arquivo de configuração na aplicação para armazenar as informações da conexão.

É necessário verificar se o servidor está aceitando conexão de outras interfaces de rede além da interface de loopback. Altere a seguinte linha no arquivo de configuração se necessário:

postgresql.conf:

```
listen_addresses = '*'
```

Usualmente, o arquivo de configuração é armazenado no diretório do banco de dados. Por exemplo:

/var/lib/pgsql/data

PostgreSQL

É possível utilizar **shared preferences** para armazenar as configurações do banco dados.

Altere o arquivo **pubspec.yaml** para incluir a dependência de **preferences**:

```
dependencies:
  preferences: ^5.1.0
  postgresql2:
  flutter:
    sdk: flutter
```

Instale o pacote adicionado nas dependências.

Inclua a importação do pacote **preferences** no arquivo **main.dart**:

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

PostgreSQL

Altere a funçao main no arquivo main.dart:

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await PrefService.init(prefix: 'pref_');
  PrefService.setDefaultValues({'host': '10.0.2.2'});
  PrefService.setDefaultValues({'port': '5432'});
  PrefService.setDefaultValues({'database': 'mobile'});
  PrefService.setDefaultValues({'user': 'mobile'});
  PrefService.setDefaultValues({'password': 'mobile'});

  runApp(MyApp());
}
```

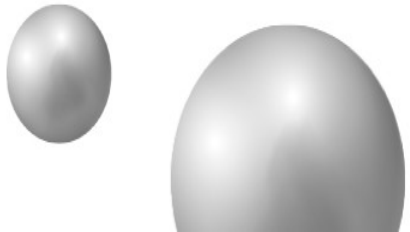
PostgreSQL

Crie o arquivo configuration.dart no diretório lib/pages:

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

class Configuration extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Configuracao"),
      ),
      body: PreferencePage([
        PreferenceTitle('DataBase'),
        TextFieldPreference(
          'Host',
          'host',
        ),
        TextFieldPreference(
          'Port',
          'port',
        ),
```

PostgreSQL

```
        TextFieldPreference(
          'Database',
          'database',
        ),
        TextFieldPreference(
          'User',
          'user',
        ),
        TextFieldPreference(
          'Password',
          'password',
        ),
      ]),
    );
  }
}
```



PostgreSQL

Inclua a importação do pacote **preferences** no arquivo **database.dart**:

```
import 'package:postgresql2/postgresql.dart';
import 'package:preferences/preferences.dart';
```

Altere a uri de conexão no arquivo **database.dart**:

```
  static String uri;
```

Inclua o método para atribuir a uri de conexão no arquivo **database.dart**:

```
  static _setUri() {
    uri = 'postgres://'+
        PrefService.getString('user').trim()+':'+
        PrefService.getString('password').trim()+'@'+
        PrefService.getString('host').trim()+':'+
        PrefService.getString('port').trim()+'/'+
        PrefService.getString('database').trim();
  }
```

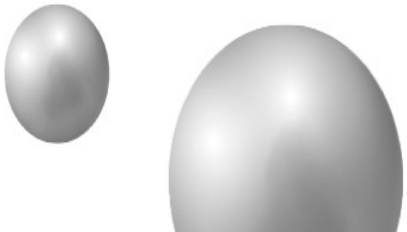
PostgreSQL

Altere o método para inicializar o acesso ao banco de dados no arquivo database.dart:

```
  static Future<bool> initDb() async {
    if ( _conn == null ) {
      String _errorMsg;
      String _responseMsg;
      print("Conectando");
      _setUri();
      var _response = await connect(uri).catchError((_error)
{
        _responseMsg = _error.message;
      });
      if ( _conn == null ) {
        if ( _response == null )
          _errorMsg = _responseMsg;
        else
          _conn = _response;
      }
```

PostgreSQL

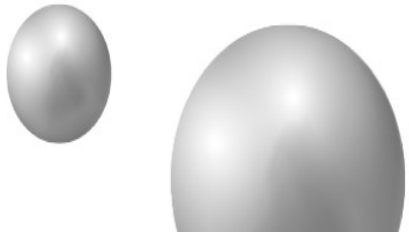
```
    if ( _conn == null )
      return Future<bool>.error(_errorMsg);
    else
      print("Conectado");
  }
  return true;
}
```



PostgreSQL

Inclua as importações do pacote preferences e da página de configuração no arquivo home.dart:

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

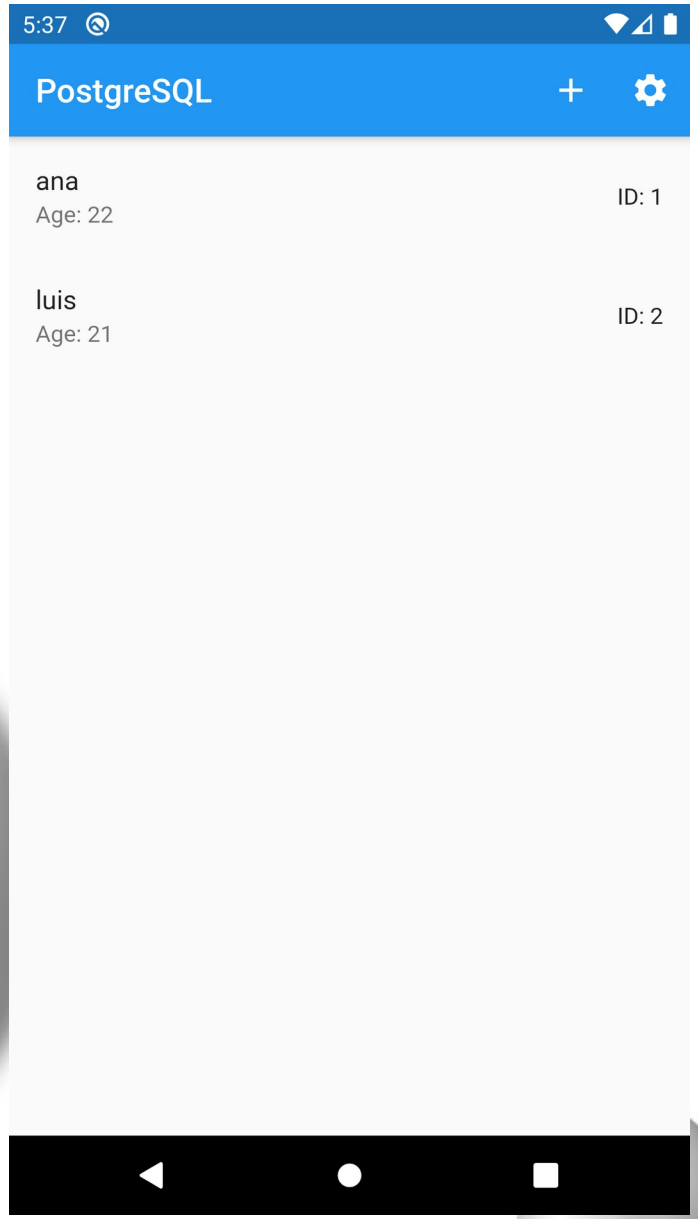


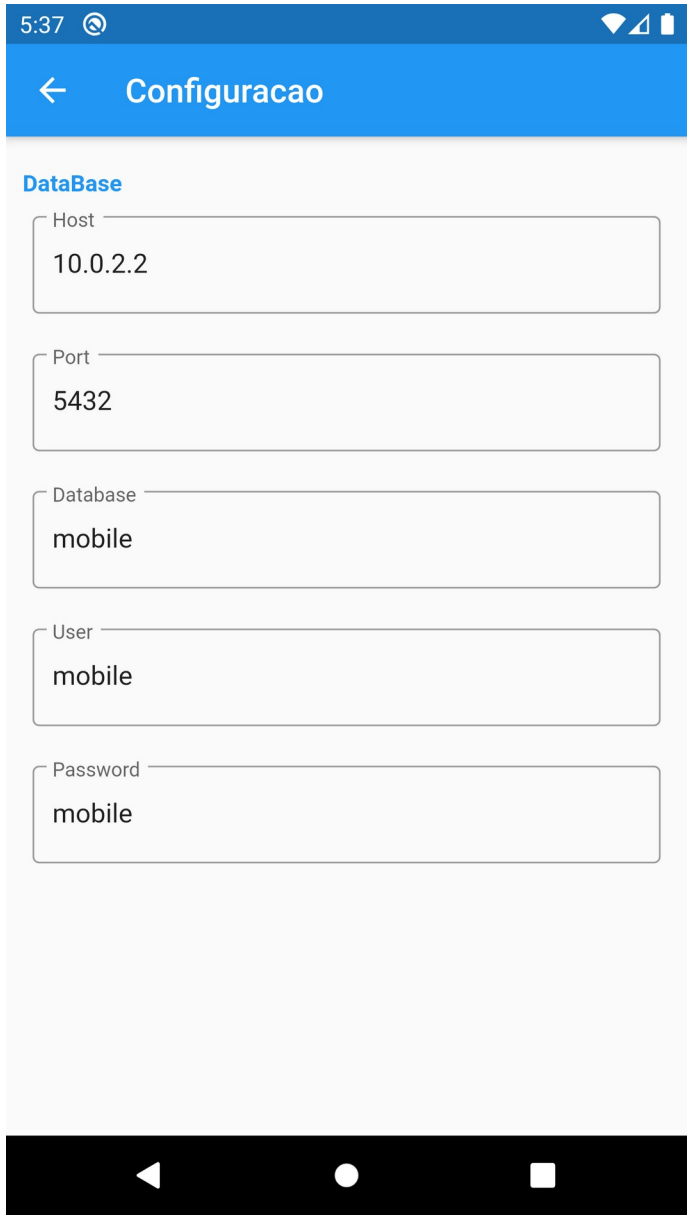
PostgreSQL

Inclua o botão de acesso à página de configuração no arquivo home.dart:

```
            IconButton(
              icon: Icon(Icons.add),
              tooltip: 'Inserir',
              onPressed: () => _detail(context, null),
            ),
            IconButton(
              icon: Icon(Icons.settings),
              tooltip: 'Configuracao',
              onPressed: () async {
                await Navigator.push(context,
                  MaterialPageRoute(builder: (context) =>
Configuration()),
                );
                setState() {};
              },
            ),
```

PostgreSQL


5:37
PostgreSQL
ana
Age: 22
ID: 1
luis
Age: 21
ID: 2


5:37
Configuracao
DataBase
Host
10.0.2.2
Port
5432
Database
mobile
User
mobile
Password
mobile

MySQL

Para a aplicação utilizando MySQL será necessário criar um usuário e um banco de dados e a tabela para a aplicação. No exemplo será utilizado o banco de dados mobile, usuário mobile e senha mobile:

```
> mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 13
Server version: 10.4.12-MariaDB MariaDB package

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and
others.

Type 'help;' or '\h' for help. Type '\c' to clear the
current input statement.

MariaDB [(none)]> CREATE DATABASE mobile;
Query OK, 1 row affected (0.000 sec)
```

MySQL

```
MariaDB [(none)]> GRANT ALL ON mobile.* TO 'mobile'@'%'
IDENTIFIED BY 'mobile';
Query OK, 0 rows affected (0.012 sec)

MariaDB [(none)]> GRANT ALL ON mobile.* TO
'mobile'@'localhost' IDENTIFIED BY 'mobile';
Query OK, 0 rows affected (0.007 sec)

MariaDB [(none)]> USE mobile;
Database changed
MariaDB [mobile]> CREATE TABLE student (
                  id      INTEGER PRIMARY KEY,
                  name    VARCHAR(255),
                  age     INTEGER,
                  email   VARCHAR(255)
                  );
Query OK, 0 rows affected (0.292 sec)
```

MySQL

É necessário verificar se o servidor está aceitando conexão de outras interfaces de rede além da interface de loopback. Comente a seguinte linha no arquivo de configuração se necessário:

my.cnf:

```
#bind-address    = 127.0.0.1
```

Usualmente, o arquivo de configuração é armazenado no diretório `/etc`.

MySQL

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

Altere o arquivo **pubspec.yaml** para incluir as dependências de **mysql1** e **preferences**:

```
dependencies:
  mysql1:
  preferences: ^5.1.0
  flutter:
    sdk: flutter
```

Instale os pacotes adicionados nas dependências.

MySQL

Copie toda a estrutura de subdiretórios e arquivos do subdiretório **lib** da aplicação **persistencia_postgresql**.

Altere a porta de conexão do banco de dados no arquivo **main.dart** no subdiretório **lib**:

```
PrefService.setDefaultValues({'host': '10.0.2.2'});
PrefService.setDefaultValues({'port': '3306'});
```

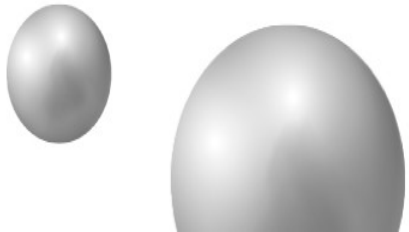
Altere o nome da aplicação no arquivo **home.dart** no subdiretório **lib/pages**:

```
appBar: AppBar(
  title: Text("MySQL"),
```

MySQL

Altere o arquivo database.dart no subdiretório lib/database:

```
import 'package:mysql1/mysql1.dart';
import 'package:preferences/preferences.dart';

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static MySqlConnection _conn;
```



MySQL

```
  static Future<bool> initDb() async {
    if ( _conn == null ) {
      String _errorMsg;
      String _responseMsg;
      print("Conectando");
      var settings = new ConnectionSettings(
        host: PrefService.getString('host').trim(),
        port:
int.parse(PrefService.getString('port').trim()),
        user: PrefService.getString('user').trim(),
        password: PrefService.getString('password').trim(),
        db: PrefService.getString('database').trim()
      );
      var _response = await
MySqlConnection.connect(settings).catchError((_error) {
        print("Erro ${_error}");
        _responseMsg = '${_error}';
      });
```

MySQL

```
    if ( _conn == null ) {
      if ( _response == null )
        _errorMsg = _responseMsg;
      else
        _conn = _response;
    }
    if ( _conn == null )
      return Future<bool>.error(_errorMsg);
    else
      print("Conectado");
  }
  return true;
}
```

MySQL

```
  static Future<List<Map>> getAll(String _table) async {
    print("Database getAll");
    List<Map> ret = List<Map>();
    String _errorMsg = "";
    String _sql = "SELECT * FROM ${_table}";
    var _rows = await _conn.query(_sql).catchError((_error)
{
      _errorMsg = '${_error}';
    });
    if (_errorMsg == "") {
      for (Row _row in _rows)
        ret.add(_row.fields);
      return ret;
    } else
      return Future<List<Map>>.error(_errorMsg);
  }
```

MySQL

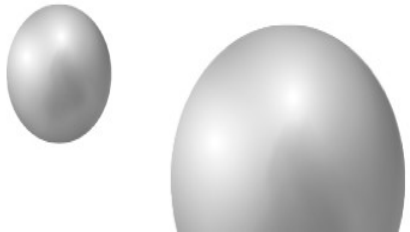
```
  static Future<Map> getByID(String _table, int _id) async {
    print("Database getByID");
    Map ret;
    String _errorMsg = "";
    String _sql = "SELECT * FROM ${_table} WHERE id='$
{_id}'";
    var _rows = await _conn.query(_sql).catchError((_error)
{
      _errorMsg = '${_error}';
    });
    if ( _rows != null ) {
      if (_rows.isNotEmpty)
        ret = _rows.first.fields;
      else
        ret = Map();
      return ret;
    } else
      return Future<Map>.error(_errorMsg);
    return ret;
  }
```

MySQL

```
  static Future<int> insert(String _table, Map _map) async {
    print("Database insert");
    int ret;
    String _errorMsg = "";
    String _sql = "SELECT COALESCE(MAX(id),0)+1 as id FROM $
{_table}";
    var _rows = await _conn.query(
      _sql
    ).catchError((_error) {
      _errorMsg = '${_error}';
    });
    if ( _rows != null ) {
      _map['id'] = _rows.first[0];
      String _fields = "";
      String _values = "";
      _map.forEach((_key, _value) {
        if (_fields != "")
          _fields += ",";
        _fields += _key;
        if (_values != "")
          _values += ",";
        _values += "'${_value}'"; });
```

MySQL

```
      String _sql = "INSERT INTO ${_table} (${_fields})
VALUES (${_values})";
      var _response = await
_conn.query(_sql).catchError((_error) {
        _errorMsg = '${_error}';
      });
      if ( _response != null )
        ret = 1;
    }
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

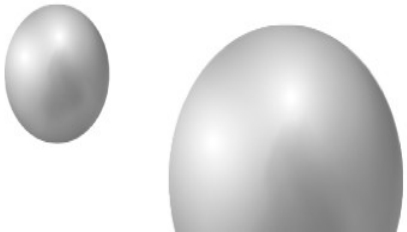


MySQL

```
  static Future<int> update(String _table, Map _map) async {
    print("Database update");
    int ret;
    String _errorMsg = "";
    String _fields = "";
    _map.forEach((_key, _value) {
      if (_key != "id") {
        if (_fields != "")
          _fields += ",";
        _fields += "${_key}='${_value}'";
      }
    });
    String _sql = "UPDATE ${_table} SET ${_fields} WHERE
id='${_map['id']}'";
    var _response = await
_conn.query(_sql).catchError((_error) {
      _errorMsg = '${_error}';
    });
```

MySQL

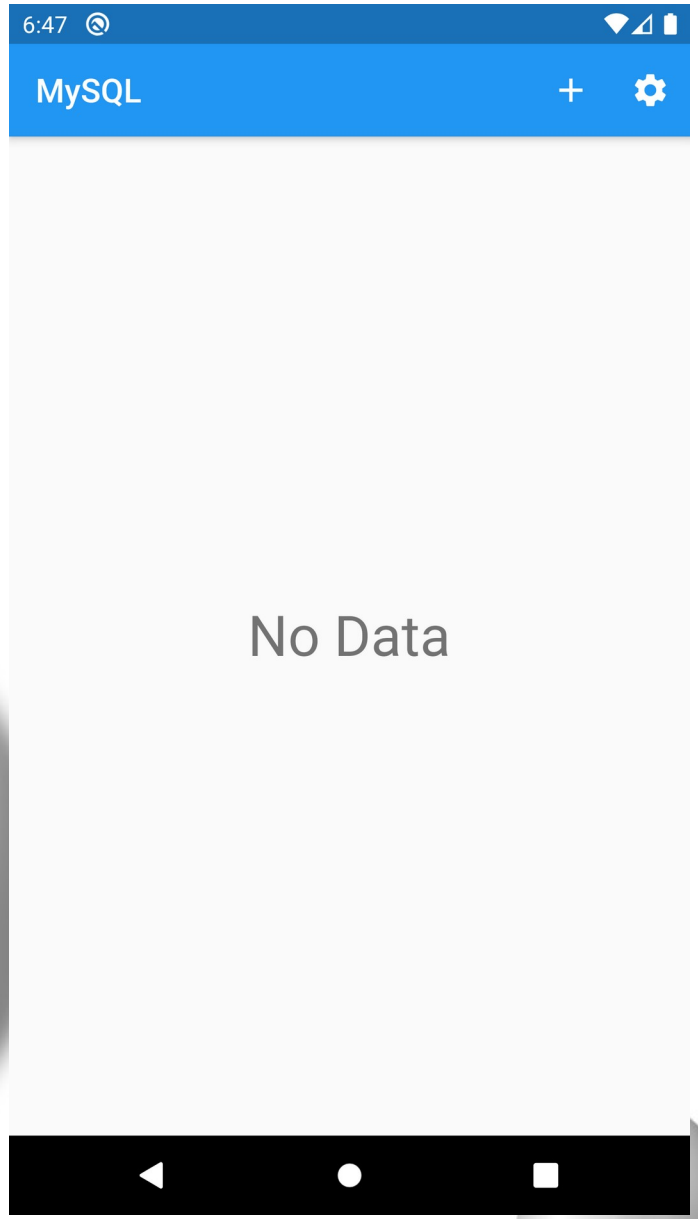
```
  if ( _response != null )
    ret = 1;
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_errorMsg);
}
```



MySQL

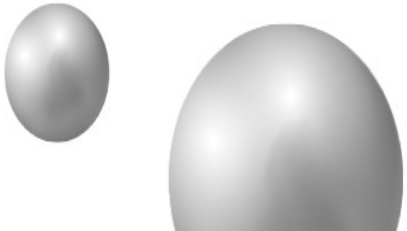
```
  static Future<int> delete(String _table, int _id) async {
    print("Database delete");
    int ret;
    String _errorMsg = "";
    String _sql = "DELETE FROM "+_table+" WHERE id='$
{_id}'";
    var _response = await
_conn.query(_sql).catchError((_error) {
      _errorMsg = '${_error}';
    });
    if ( _response != null ) ret = 1;
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }

  void close() {
    print("CloseDB");
    _conn.close();
  }
}
```

MySQL



Local

É possível utilizar arquivos texto locais para a persistência dos dados da aplicação. A manipulação de arquivos texto sem um engine para otimizar as operações de pesquisa e alteração dos dados não é recomendada devido a baixa performance ou a necessidade de implementar rotinas complexas para executar essas operações de modo eficiente. Porém para um pequeno volume de dados, o uso de arquivos locais pode ser uma alternativa viável.



Local

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

Altere o arquivo **pubspec.yaml** para incluir a dependência de **path_provider** :

```
dependencies:
  path_provider: ^1.1.0
  flutter:
    sdk: flutter
```

Instale o pacote adicionado nas dependências.

Local

Copie toda a estrutura de subdiretórios e arquivos do subdiretório **lib** da aplicação **persistencia_sqlite**.

Altere o nome da aplicação no arquivo **home.dart** no subdiretório **lib/pages**:

```
appBar: AppBar(
  title: Text("Local"),
```

Local

Altere o arquivo student_dao.dart no subdiretório lib/dao para remover as referência ao nome da tabela:

```
Future<List<Student>> getAll() async {
  print("DAO getAll");
  _errorMsg = "";
  List<Map> _maps = await DatabaseHelper.getAll()
    .catchError((_error) {_errorMsg = _error;});

Future<Student> getByID(int _id) async {
  print("DAO getByID");
  Student ret;
  _errorMsg = "";
  var response = await DatabaseHelper.getByID(_id)
    .catchError((_error) {_errorMsg = _error;});

Future<int> insert(Student _student) async {
  print("DAO insert");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.insert(_student.toJson())
    .catchError((_error) {_errorMsg = _error;});
```

Local

```
Future<int> update(Student _student) async {
  print("DAO update");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.update(_student.toJson())
    .catchError((_error) {_errorMsg = _error;});

Future<int> delete(int _id) async {
  print("DAO delete");
  int ret;
  _errorMsg = "";
  ret = await DatabaseHelper.delete(_id)
    .catchError((_error) {_errorMsg = _error;});
```

Local

Altere o arquivo database.dart no subdiretório lib/database:

```
import 'package:path_provider/path_provider.dart';
import 'dart:convert';
import 'dart:io';

class DatabaseHelper {
  static final DatabaseHelper _instance =
DatabaseHelper.internal();
  factory DatabaseHelper() => _instance;
  DatabaseHelper.internal();

  static List<Map> _list;
  static int _maxId = 0;

  static Future<String> get _localPath async {
    final directory = await
getApplicationDocumentsDirectory();
    return directory.path;
  }
```

Local

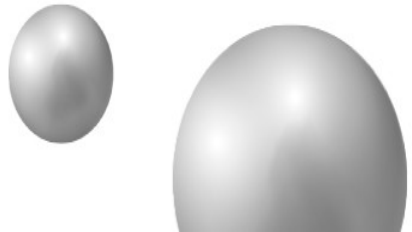
```
static Future<File> get _localFile async {
  final path = await _localPath;
  return File('$path/local_persistence.json');
}
```

Local

```
static Future<bool> initDb() async {
  if ( _list == null ) {
    print("InitDB");
    String _errorMsg = "";
    try {
      final file = await _localFile;
      if (!file.existsSync()) {
        _list = List<Map>();
      }
      else {
        String _contents = await file.readAsString();
        print("Content; [$_contents]");
        _list = List<Map>();
        final _data = json.decode(_contents);
        _list = List<Map>.from(
          _data["data"].map((x) {
            if (x['id'] > _maxId)
              _maxId = x['id'];
            return x;
          })
        );
      }
```

Local

```
    } catch (_error) {
      _errorMsg = _error;
    }
    if ( _list == null )
      return Future<bool>.error(_errorMsg);
  }
  return true;
}
```



Local

```
  static Future<File> write() async {
    final file = await _localFile;
    Map<String, dynamic> _map = {
      "data": List<dynamic>.from(_list.map((x) => x)),
    };
    String _json = json.encode(_map);
    print("Contents: [$_json]");
    return file.writeAsString('$_json');
  }

  static Future<List<Map>> getAll() async {
    print("Database getAll");
    if (_list != null)
      return _list;
    else
      return Future<List<Map>>.error("Erro carregando
dados");
  }
```

Local

```
static Future<Map> getByID(int _id) async {
  print("Database getByID");
  Map ret = Map();
  if (_list != null) {
    int _index = _list.indexWhere((e) => e['id'] == _id);
    if (_index >= 0)
      ret = _list[_index];
  }
  return ret;
}
```

Local

```
static Future<int> insert(Map _map) async {
  print("Database insert");
  int ret;
  String _errorMsg = "";
  if (_list != null) {
    int _newId = _maxId+1;
    _map['id'] = _newId;
    _list.add(_map);
    await write();
    _maxId = _newId;
    ret = 1;
  } else
    _errorMsg = "Dados nao carregados";
  if (ret != null)
    return ret;
  else
    return Future<int>.error(_errorMsg);
}
```

Local

```
  static Future<int> update(Map _map) async {
    print("Database update");
    int ret;
    String _errorMsg = "";
    if (_list != null) {
      int _index = _list.indexWhere((e) => e['id'] ==
_map['id']);
      if (_index >= 0) {
        _list[_index] = _map;
        await write();
        ret = 1;
      } else
        ret = 0;
    } else
      _errorMsg = "Dados nao carregados";
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }
```

Local

```
  static Future<int> delete(int _id) async {
    print("Database delete");
    int ret;
    String _errorMsg = "";
    if (_list != null) {
      int _index = _list.indexWhere((e) => e['id'] == _id);
      if (_index >= 0) {
        _list.removeAt(_index);
        await write();
        ret = 1;
      } else ret = 0;
    } else _errorMsg = "Dados nao carregados";
    if (ret != null)
      return ret;
    else
      return Future<int>.error(_errorMsg);
  }

  void close() {
    print("CloseDB");
  }
}
```

Local