Integração das ferramentas

→ Cria-se a pasta *util* e dentro dela o arquivo *requests.dart*

//////// NO ARQUIVO **requests*.dart***

Importamos as bibliotecas ‘dart:async’, ‘dart:convert’ e ‘http’.

A biblioteca ‘dart:async’ nos permite utilizarmos a programação assíncrona suportada pela linguagem Dart para tratar de funções ou eventos que possam consumir longos tempos de execução, o que faria com que a aplicação bloqueasse (ou melhor dizendo, “travasse”) caso fossem síncronas. No nosso caso, obter os dados do servidor pode demorar, dessa forma essas operações remotas são assíncronas para que a aplicação continue funcionando mesmo que a execução das chamadas do banco de dados dure algum tempo.

A biblioteca ‘dart:convert’ traz métodos de conversão de dados. No nosso app, ela permite a serialização e desserialização dos objetos/strings no formato **JSON** que recebemos da ***API rest*** do banco de dados.

A biblioteca ‘http’ nos permite utilizar métodos de requisição HTTP, que são os métodos GET e POST utilizados pelo aplicativo para obtenção e inserção de dados, respectivamente.

```dart
import 'dart:async';
import 'dart:convert';
import 'package:http/http.dart' as http;

// final url = 'http://localhost:8000';
final url = 'http://10.0.2.2:8000';
```

A string “url” é uma constate de tempo de execução que define o *host* e a porta do servidor onde as rotas da API de comunicação estão localizadas. . Utilizou-se a url “10.0.2.2” pois é a url que o emulador consegue se comunicar com o localhost.

Criamos a função ***getContactList*** que busca e retorna a lista dos contatos registrados no banco de dados. Ela é uma função assíncrona, então retorna um ***Future<List>***, isto é, futuramente ele retornará uma lista (*List*), pois as operações remotas de banco de dados podem levar um tempo considerável para se completarem e dessa forma a função assíncrona retorna inicialmente um objeto "Future" que é um objeto incompleto, que futuramente será 'completado' com valores do tipo de dados esperado (nesse caso, um *List*). Para criar uma função ou método assíncronos, é necessário que ele retorne um ***Future*** e que seja marcado como ***async***.

```
Future<List> getContactList() async {
  try {
    var getAllContacts =
        await http.get("$url/contacts/", headers: {
      "Content-Type": "application/json",
    });

    var contactList = json.decode(getAllContacts.body);
    print(getAllContacts.body);

    return contactList as List;
  } catch (e) {
    print("O erro é: $e");
    return null;
  }
}
```

Fazemos a requisição com o método *HTTP* GET (***http.get***) passando como parametros a ***url*** do servidor do banco de dados e a rota da *API rest* "***/contacts***", que fornece a lista dos contatos do banco (nota-se que é necessário especificar nas headers do GET o tipo do conteúdo que está sendo requisitado, no caso: *application/json*) e por fim salvamos a resposta do servidor na variável ***getAllContacts*** que recebe o tipo ***Response***.

Utilizamos o codec ***json*** da biblioteca 'dart:convert' (que é um objeto com métodos de codificação e decodificação) para decodificador o ***body*** da resposta (*Response*) do servidor (*body* que contém a lista de contatos) e guardamos na variável ***contactList*** e que por fim é o retorno da função (*as List* é como se faz um "*cast*" para uma List em Dart).

Criamos também a função ***createNew*** que registra o contato no banco de dados. Recebe os parametros ***name***, ***email*** e ***phone*** (todos String) que são as informações de cada contato. Também é uma função assíncrona e retorna um ***Future<void>*** que não se completa (no fim não retorna nada).

```
Future<void> createNew(String name, String email, String phone) async {
  var payload = {
    "name": name,
    "email": email,
    "phone": phone,
  };

  var body = json.encode(payload);
  var post = await http.post('$url/contact/new/',
      headers: {"Content-Type": "application/json"}, body: body);

  if (post.statusCode == 200)
    print('Contact created → ${post.statusCode}');
  else
    print('Error while creating new contact → ${post.statusCode}');
}
```

A variável ***payload*** é um Map, uma estrutura de dados com chaves (*keys*) que atribuem a valores (*values*), por exemplo: '*map = {"nome": "Fulano", "idade": 23}',* onde os valores podem ser acessados pela referencia das chaves, por exemplo: *'print(map["nome"]); // printa Fulano'*.

O Map ***payload*** contém o formato dos dados semelhantemente ao formato de envio e recebimento de dados, **JSON**. Assim, serializamos o *payload* em uma *String* no formato JSON e o enviamos pelo método *HTTP* POST (***http.post***) passando como parametros o servidor e a rota da *API rest* que permite a inserção de dados no banco, "***/contact/new***". Se a requisição ocorrer bem, ou seja, retornar o *status* 200 (*OK*) o contato foi registrado no banco de dados.

Também criamos a função ***getContact*** que retorna um contato específico buscado pelo seu ***id***. Ela retorna um Map, que é a representação JSON do contato, isto é, um Map contendo as suas informações associadas por *name*, *email* e *phone*.

```
Future<Map> getContact(int id) async {
  var getContact = await http
      .get('$url/contact/$id', headers: {"Content-Type": "application/json"});
  var contact = json.decode(getContact.body);
  print(getContact.body);

  return contact;
}
```

//////// NO ARQUIVO ***list.dart***

Importamos o arquivo ***requests.dart*** e damos o nome de "**db**".

```
import 'package:flutter/material.dart';
import 'package:listadecontatos/models/contact.dart';
import 'package:listadecontatos/widgets/details.dart';
import 'package:listadecontatos/util/requests.dart' as db;
```

Substituímos as inserções de contatos falsos de teste para chamar o método ***populateList*** no construtor do ***ContactState***.

```
class ContactState extends State<ContactList> {
  List<Contact> _contacts = List<Contact>();

  ContactState() {
    // _contacts.add(Contact("Teste 1", "12345678", "teste1@teste.com"));
    // _contacts.add(Contact("Teste 2", "5242364", "teste2@teste.com"));
    // _contacts.add(Contact("Teste 3", "98521452", "teste3@teste.com"));

    populateList();
  }
```

O método ***populateList*** inicializa a lista de contatos ***_contacts*** com a nova lista copiada do banco de dados.

```
populateList() async {
  var list = await db.getContactList();
  var contacts = List<Contact>();

  for(var c in list) {
    contacts.add(Contact(c["name"], c["phone"], c["email"]));
    print('nome: ${c["name"]}');
  }

  setState(() {
    if((contacts?.length ?? 0) > 0) _contacts = contacts;
    else _contacts = List<Contact>()..add(Contact("null", "null", "null"));
  });
}
```

Obtemos a lista de contatos do banco de dados com o método ***getContactList*** que criamos no arquivo *requests.dart* e nomeamos de *db* (*var list = await db.getContactList()*). Usamos a palavra-chave da linguagem Dart, ***await***, para esperarmos a execução de uma função assíncrona.

Iteramos a lista de contatos do banco e adicionamos cada contato da lista do banco numa lista temporária: ***contacts***.
Assim, para modificar os valores de um atributo ou variável de uma classe que é um ***State***, no Flutter, utilizamos o método ***setState***.
Dentro do *setState* verificamos se a lista de contatos do banco não estava vazia (*length* > 0) e se não estiver, a lista ***_contacts*** é atribuída com os valores da lista do banco de dados. Se estiver vazia, cria-se uma lista com os atriutos *"null"*.

//////// NO ARQUIVO ***details.dart***

Importamos o arquivo ***requests.dart*** que contém as funções de obtençao e criamos de contatos no banco e o nomeamos de "***db***".

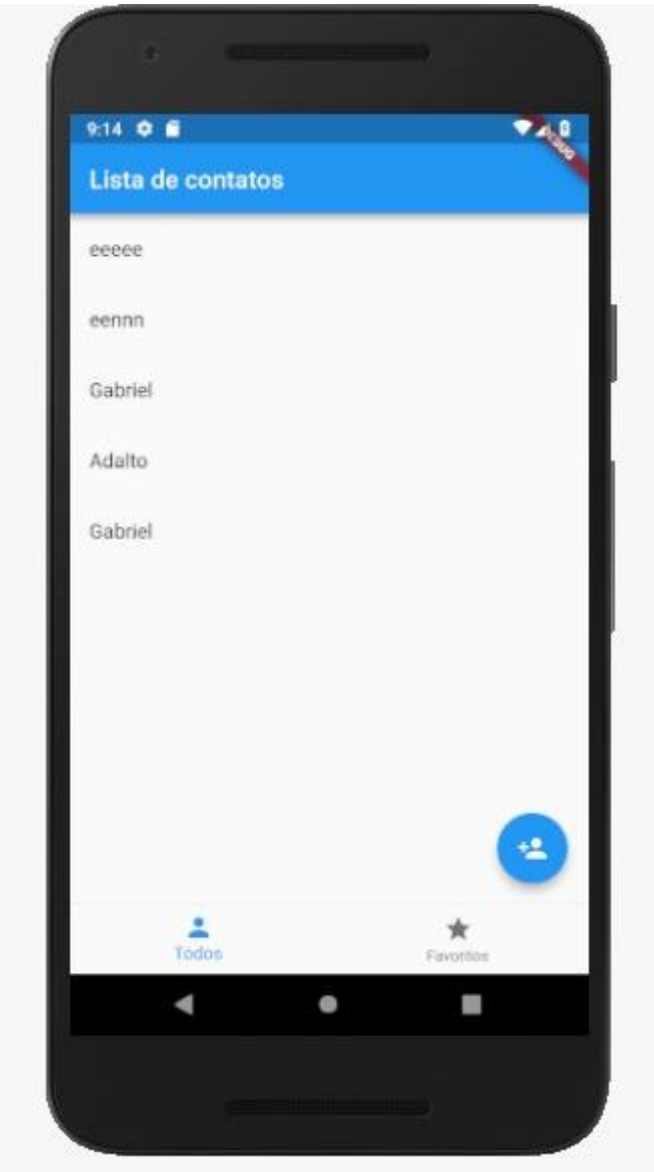
```dart
import 'package:flutter/material.dart';
import 'package:listadecontatos/models/contact.dart';
import 'package:listadecontatos/util/requests.dart' as db;
```

Atualizamos o método ***_addContact*** inserindo as linhas "***await db.createNew(_contact.name, _contact.email, _contact.tel)***" para enviarmos ao banco de dados os novos contatos salvos.

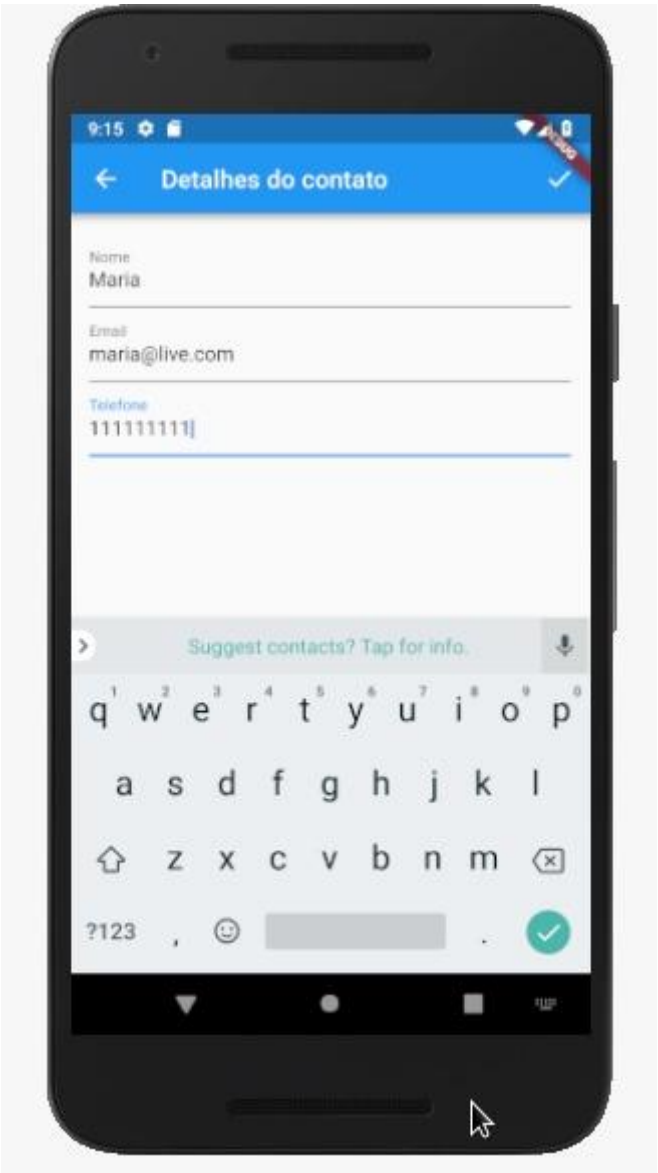
```dart
void _addContact() async {
    if(_contact == null) {
        _contact = Contact(_controllerName.text, _controllerName.text, _controllerEmail.text);
        await db.createNew(_contact.name, _contact.email, _contact.tel);
    } else {
        _contact.name = _controllerName.text;
        _contact.email = _controllerEmail.text;
        _contact.tel = _controllerTel.text;
        await db.createNew(_contact.name, _contact.email, _contact.tel);
    }

    print("created.");
}
```

/// **Alguns prints dos resultados**

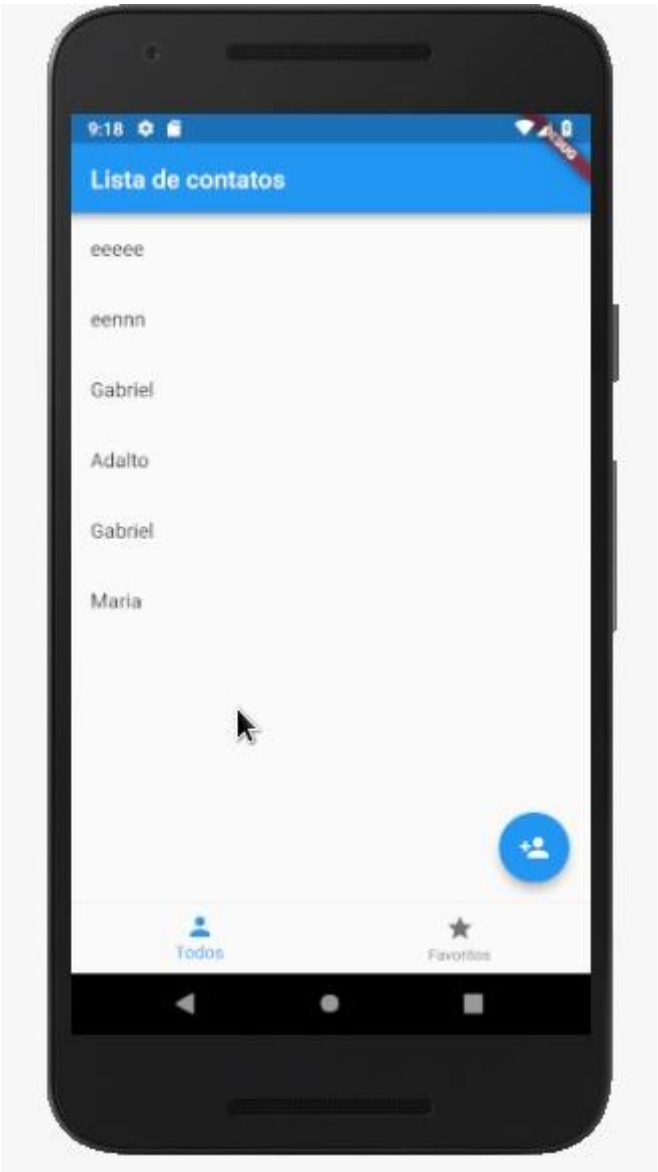
Lista de contatos quando se inicia o app:



Criação e armazenamento de um novo contato (Maria):



Lista completa depois de salvar o contato (Maria):



Dados printados no terminal:

```
Performing hot restart...
Restarted application in 1,805ms.
I/flutter ( 9234): [{"name": "eeeee", "email": "1@adsd.com", "phone": "eeeee"}, {"name": "eennn", "email": "1@adsd.com", "phone":
"eeeee"}, {"name": "Gabriel", "email": "asdlfkh@laksjfh.com", "phone": "Gabriel"}, {"name": "Adalto", "email": "a98sdlfkh@laksjfh.
com", "phone": "124890908"}, {"name": "Gabriel", "email": "asdlfkh@laksjfh.com", "phone": "000000000000"}, {"name": "Maria", "emai
l": "maria@asd.com", "phone": "Maria"}]
I/flutter ( 9234): nome: eeeee
I/flutter ( 9234): nome: eennn
I/flutter ( 9234): nome: Gabriel
I/flutter ( 9234): nome: Adalto
I/flutter ( 9234): nome: Gabriel
I/flutter ( 9234): nome: Maria
```