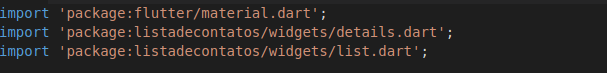
Layout da Agenda (App Lista de contatos)

//////// NO ARQUIVO ***main.dart***

Antes de tudo, importa-se o **material.dart** que é a classe que o flutter usa para os Widgets

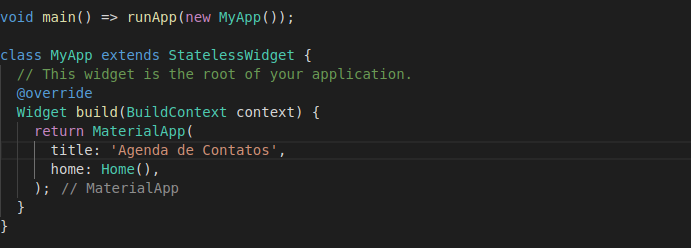
Também tem os outros imports das outras classes que a gente vai criar

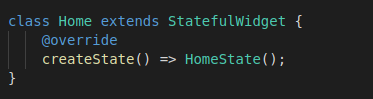


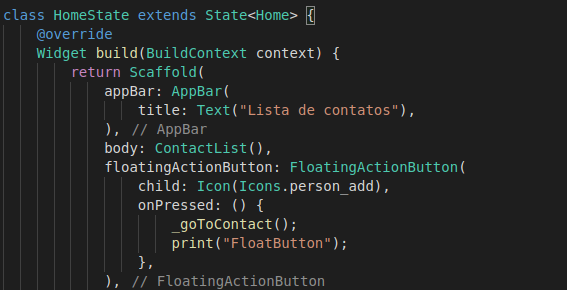
Em Dart a função main( ) é onde o programa vai rodar: **void main( ) => runApp(MyApp());**

é a função main chamando o construtor do MyApp(). Note que usei o formato de “fat arrow”, que é quando se chama uma expressão ou função/metodo que ocupa apenas uma linha com o “=>”.

A classe MyApp herda de StatelessWidget, ou seja, ela própria é uma StatelessWidget, que é uma Widget que não muda de estado. O método que ela herda é o **build** que deve ser sobreescrito (@override) da seguinte forma:

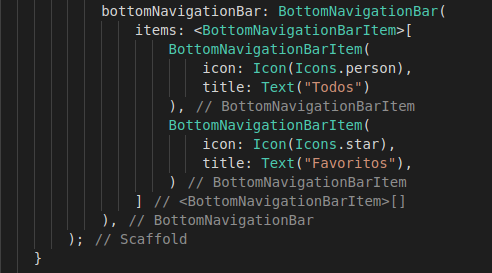
Retornando um MaterialApp contendo o título do App e setando a página inicial (home).

Logo após, criamos a classe Home que é chamada no MaterialApp como página inicial. Ela é uma StatefulWidget, ou seja, seu estado é mutável, e é um State (estado) que o muda. Por isso, herda o método **createState** que chama o State de Home (HomeState).

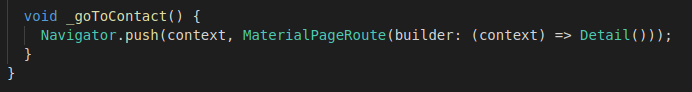
A classe HomeState herda de State<Home>, isto é, é um estado da classe Home.

Herda o método build que constrói o app. Ela retorna um **Scaffold**, que é um Layout básico. O Scaffold contém o **AppBar**, que é a barra superior do aplicativo, contendo o título da página atual e, em alguns casos, alguns ícones e botões.

Assim, o Scaffold também possui o parâmetro body, que é o corpo do layout em si. No nosso tutorial, chamamos **ContactList()**, construtor da classe que criaremos mais adiante para listar os contatos. FloatingActionButton é o botão “flutuante” que tem a função de adicionar contatos. É possível setar ícones para os botões com os materiais Icon e Icons. No parametro onPressed é passada a página do app em que será redirecionado (\_goToContact()) faz isso).

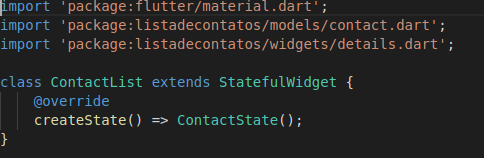
Ainda na construção do Scaffold, criamos um BottomNavigationBar com dois Items. (Uma barra de navegação inferior, abaixo do body do Scaffold).

Criamos o método **\_goToContact()** que nos redireciona para a página de detalhes de contato que criaremos logo à frente no tutorial. Ele usa o “stack” de páginas Navigator empurrando a classe Detail.



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

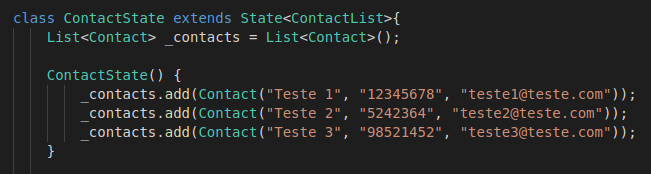
Criamos então uma nova classe chamada **ContactList**, que listará os contatos no body do Scaffold da página inicial. Ela é uma StatefulWidget, então seu estado irá variar.



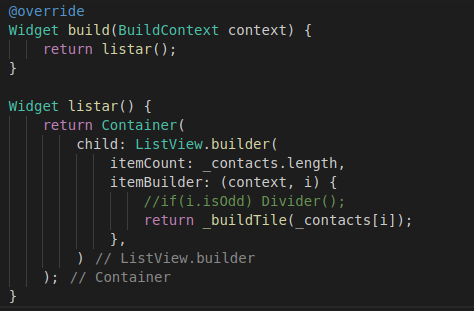
Criamos então o estado **ContactState** da nossa ContactList. (State<ContactList> diz que ele é um estado de ContactList).

Criamos uma lista de contatos, uma estrutura de dados que irá armazenar temporariamente e em tempo de execução os contatos existentes e adicionados. Então, no construtor de ContactState nós populamos, apenas para testes, algumas contatos (A classe **Contact** também será criada mais à frente, ela é um model, ou seja, uma classe que serve como tipo de objetopara nosso app).

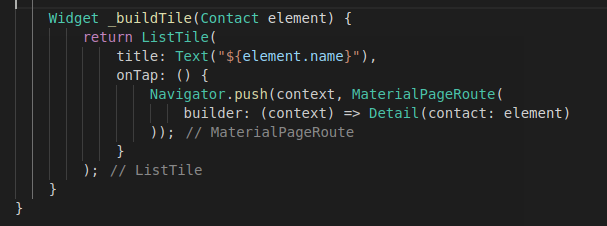
Note que criamos a lista **\_contacts** com um \_ (underline) na frente do nome da variável. Isso porque em Dart underlines na frente do nome das variáveis ou constantes significa que são privadas e somente esta atual classe tem acesso.



No build de ContactState chamamos o método **listar**. Este método lista (obviamente) os contatos dentro de uma **ListView**. Ele retorna um **Container** contendo como child uma ListView. Quando se programa usando Flutter, alguns materiais (componentes gráficos) possuem nenhuma, uma ou mais *child/children* que são componentes filhos e estão dentro dos componentes que os chamam. Uma forma fácil de criar uma ListView e populá-la é usando o método **builder**. Ele precisa dos dois parâmetros: **itemCount**, que conta a quantidade de itens da ListView; e **itemBuilder** que constrói os itens (chamando \_buildTile).



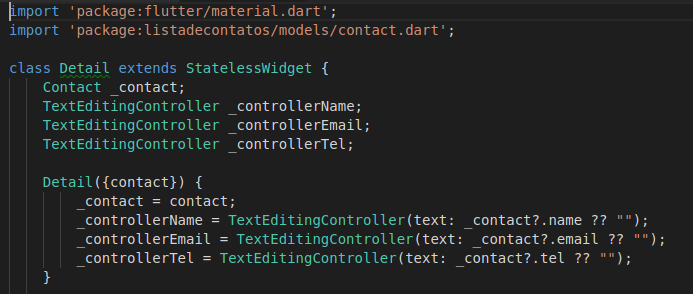
O método privado (lembre-se do \_ underline) **\_buildTile** retorna um modelo de **ListTile**, ou seja, como cada “tile” (ou contato) será feito: o título do contato e a função que ele fará quando o usuário tocar nele, **onTap.** OnTap dará um “push” na stack de navegação de páginas Navigator, dando a rota para os detalhes do contato.



//////// NO ARQUIVO ***detail.dart***

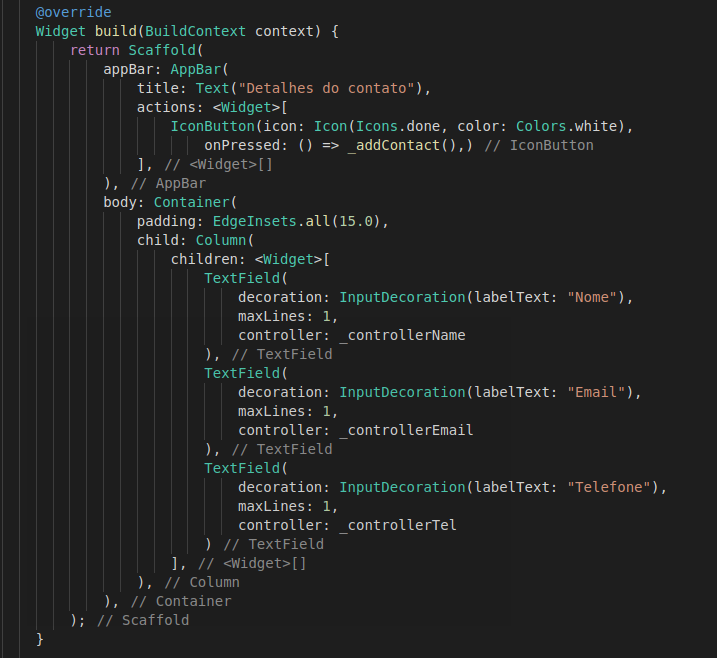
Criamos então a classe **Detail** (que é StatelessWidget). Ela nos dará os detalhes das informações de cada contato.

**TextEditingController** controla as ações de input nas caixas de texto para criação e visualização das informações do contato. No construtor da classe inicializamos os TextEditingControllers com os contatos e em cada campo com os atributos respectivos dos contatos “name”, “email” e “tel”, passando para os parametros *text* de TextEditingController as informações.

(Nota: em Dart 2 o operador “**?.”** designa uma condição e atribuição *in-line* que permite acessar os dados apenas se o objeto \_contact **não estiver null**. Se estiver, o operador “**??**” nos condiciona que, se o valor é null, então agora será **“”** (string vazia).

CONTINUAÇÃO ABAIXO (print ficou muito grande)

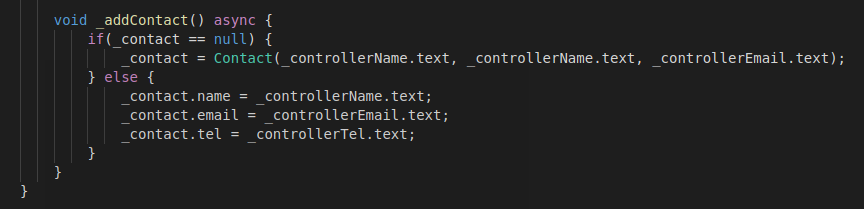
Na build de Detail, retornamos um Scaffold, construído com os componentes já vistos e da mesma forma que anteriormente. Importante notar um parametro novo dentro do AppBar: **actions**. Ele nos permite criar ações (widgets) no AppBar. No nosso caso, criamos um botão com ícone que quando pressionado aciona o método privado **\_addContact** (veremos logo à frente).

Então criamos o body com um Container contendo como child uma **Column** com vários children. Nos children são criados os **TextField**s onde serão digitados **/** mostrados os dados do contato e neles são atribuídos os controllers criados anteriormente. (Note que **padding: EdgeInsets.all(15.0)** aparece na construção do Container; é o ajuste de posicionamento).

| |

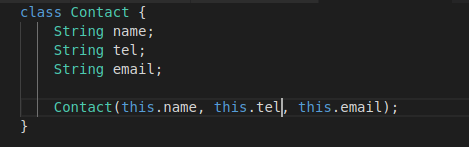
\/

Por fim, implementamos o método **\_addContact**. Há uma verificação se o contato é nulo, e se for, cria um novo contato. Se não for, edita os dados do contato já existente. (Nota: **async** denota um método assíncrono).



//////// NO ARQUIVO ***contact.dart***

E então a classe **Contact**. É a classe modelo para o objeto que representa um contato, contendo os atributos name, tel e email e um construtor que já seta os atributos.



// Atualização do tutorial: dia 20/11/2018

→ 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.



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.

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***.

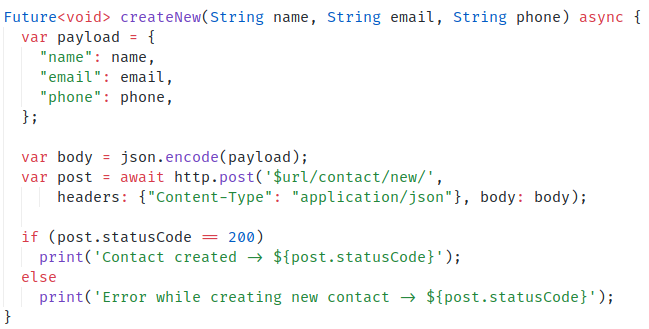


Fazemos a requisição com o método *HTTP* GET (***http.get***) passando como parametros a ***url*** do servidor do banco de dados e a rota da *API rest* “***/contacts***”, que fornece a lista dos contatos do banco (nota-se que é necessário especificar nas headers do GET o tipo do conteúdo que está sendo 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).



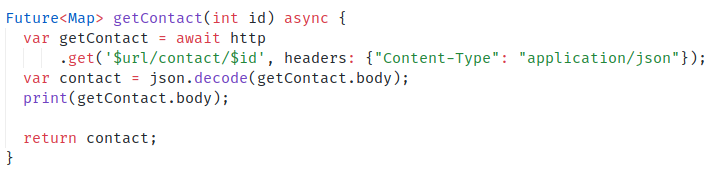
A variável ***payload*** é um Map, uma estrutura de dados com chaves (*keys*) que atribuem a valores (*values*), por exemplo: ‘*map = {“nome”: “Fulano”, “idade”: 23}’,* onde os valores podem ser acessados pela referencia das chaves, por exemplo: *‘print(map[“nome”]); // printa Fulano’*.

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.

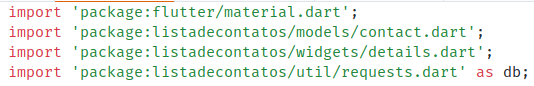
Também criamos a função ***getContact*** que retorna um contato específico buscado pelo seu ***id***.

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*.

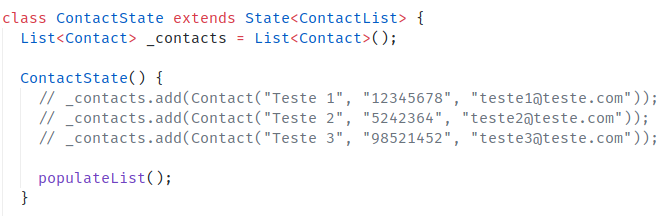


//////// NO ARQUIVO ***list.dart***

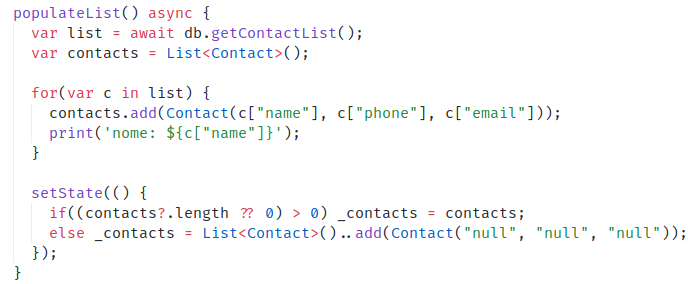
Importamos o arquivo ***requests.dart*** e damos o nome de “**db**”.

******

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



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



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***.

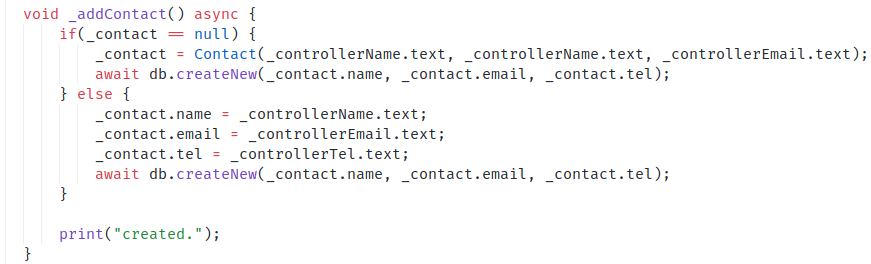
Assim, para modificar os valores de um atributo ou variável de uma classe que é um ***State***, no Flutter, utilizamos o método ***setState***.

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***

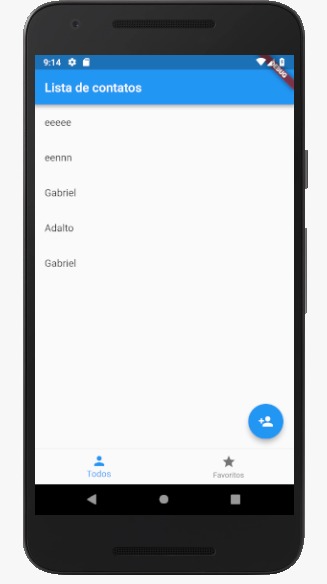
Importamos o arquivo ***requests.dart*** que contém as funções de obtençao e criamos de contatos no banco e o nomeamos de “***db***”.

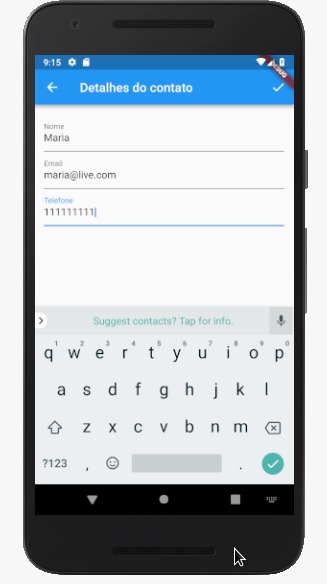


Atualizamos o método ***\_addContact*** inserindo as linhas “***await db.createNew(\_contact.name, \_contact.email, \_contact.tel)***” para enviarmos ao banco de dados os novos contatos salvos.

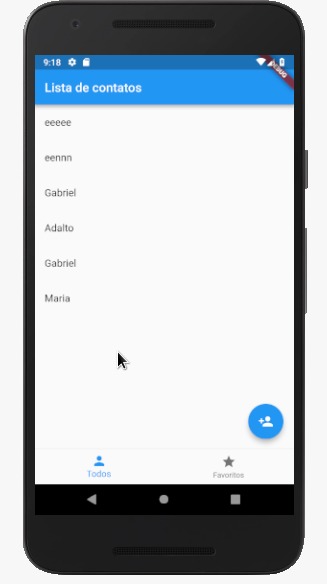
/// Alguns prints dos resultados

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:

