CENTRO UNIVERSITÁRIO FAMETRO

SISTEMAS DE INFORMAÇÃO

ERICK RAINIER DE FREITAS SEREJO

APLICATIVO CUIDA

Um novo jeito de se cuidar

FORTALEZA

2021

ERICK RAINIER DE FREITAS SEREJO

APLICATIVO CUIDA

Um novo jeito de se cuidar

Artigo TCC apresentado ao curso de Bacharel em Sistemas de Informação da Faculdade Metropolitana da Grande Fortaleza – FAMETRO – como requisito para a obtenção do grau de bacharel, sob a orientação da prof.ª Tiago Guimarães Sombra.

FORTALEZA

2021

ERICK RAINIER DE FREITAS SEREJO

APLICATIVO CUIDA

Um novo jeito de se cuidar

Artigo TCC apresentada no dia 15 de dezembro de 2021 como requisito para a obtenção do grau de bacharel em Sistemas de Informação da Faculdade Metropolitana da Grande Fortaleza – FAMETRO – tendo sido aprovado pela banca examinadora composta pelos professores abaixo:

BANCA EXAMINADORA

_____
Profº. Tiago Guimarães Sombra
Orientador – Faculdade Metropolitana da Grande Fortaleza

_____
Profº. João Leonardo Silveira Neto
1º Examinador - Faculdade Metropolitana da Grande Fortaleza

_____
Profº. Fábio Henrique Fonseca de Sousa
2º Examinador - Faculdade Metropolitana da Grande Fortaleza

Ao professor Tiago Sombra, que com sua dedicação e cuidado de mestre, orientou-me na produção deste trabalho.

AGRADECIMENTOS

A Deus pelo dom da vida, pela ajuda e proteção, pela Sua força e presença constante, e por me guiar à conclusão de mais uma preciosa etapa de minha vida. A minha filha, por ser uma força que me faz levantar todo dia a fim de mostrar a ela que sou um homem capaz. A meu segundo filho, que mesmo não tendo nascido, estará sempre em minha memória. Ao meu terceiro, que durante este projeto, ainda está no ventre de sua mãe. E a ela, minha mulher, que me mostrou que eu sou um homem agraciado por Deus.

Aplicativo CUIDA: um novo jeito de se cuidar.

RESUMO

Com a facilidade do uso de aplicativos para auxiliar a vida, tem sido necessário a reflexão diante das necessidades que podem ser atendidas com a palma da mão. Serviços e produtos que oferecem conhecimento e informação, fazem parte de um ramo de negócio rentável e atrativo. Utilizar de estudos e pesquisas que serão usadas para confortar, auxiliar e direcionar pessoas em um momento difícil, pode se tornar algo altamente requisitado. Por meio disso o aplicativo - nomeado de CUIDA - foi desenvolvido para dar a maior quantidade de informação, localização, apoio e discrição para os portadores da neoplasia maligna, também conhecida como câncer.

Palavras-chave: Aplicativo, Flutter. Informação, Serviço, Câncer.

ABSTRACT

With the ease of using applications to help life, it has been necessary to reflect on the needs that can be met with the palm of the hand. Services and products that offer knowledge and information are part of a profitable and attractive line of business. Using studies and research that will be used to comfort, help and guide people in a difficult time, can become something highly requested. Through this, the application - named CARE - was developed to give the greatest amount of information, location, support and discretion to patients with malignant neoplasm, also known as cancer.

Key words: Application, Flutter. Information, Service, Cancer.

1 INTRODUÇÃO

O uso de aplicativos que facilitam a vida é perceptível. Bancos digitais que nos oferecem serviços sem precisarmos nos deslocar até uma agência; pacotes e promoções destes que – sem necessidade de letras pequenas ou contratos de longas páginas - simplificam o que oferecem nos informando o necessário. São tratativas assim que diminuem o tempo de busca do usuário e fideliza o futuro cliente que busca pelo melhor serviço. Há aplicativos que também, para nos economizar o tempo, oferecem serviços essenciais de entrega de comida, informação e transporte.

É com aplicativos assim que uma solução mais viável para determinada problemática ou uma necessidade que precisa ser atendida, que serviços e produtos são criados. E para atender isso, um sistema de informação baseado nas necessidades a serem respondidas, com uma nova roupagem, estrutura e atendimento, que empresas surgem, startups começam e programas são criados.

Um exemplo claro disso, o aplicativo SCHOLAR (LEAL, LEAL, 2019) foi pensado nas necessidades dos universitários. Os estudantes que trabalharam neste projeto viram a carência de acessar o sistema da faculdade pelo celular já que o site não possuía uma página responsiva (adaptada para o formato da tela de um dispositivo móvel); nisso o problema que surgiu para eles, foi uma solução viável de projeto.

Diante da visualização de um problema ou carência, o motivo deste trabalho foi uma das questões mais levantadas em buscadores nos últimos anos. Informações sobre doenças sempre esteve no topo das dúvidas de usuários de rede. O famoso "estou sentindo...", "estou com..." ou até mesmo o sintoma em si. Tudo isso é alinhado com as dúvidas sobre a prevenção de uma possível doença que é sempre delicada diante do diagnóstico.

Devido a isso, campanhas de prevenção como o Outubro Rosa e Novembro Azul vem sendo difundidos em países que carecem de informação sobre a gravidade do problema que é o câncer.

Por conta disso o CUIDA foi desenhado e pensado para solucionar o problema da falta de informação sobre os devidos cuidados, precauções e exames que a pessoa deve tomar.

Tomando como exemplo prático: ao sentirmos determinados sintomas, específicos e característicos que não eram recorrentes anteriormente, a preocupação maior do indivíduo basicamente integrado com tecnologia, é acessar um buscador de melhor facilidade e informar os sintomas sentidos.

Por conta deste contexto, percebemos que a praticidade para retirar as devidas dúvidas, sobre determinada doença, é necessária. Contudo, isso não exclui em hipótese alguma a consulta médica.

Utilizar das informações catalogadas, que facilitam e reduzem nossa ansiedade diante de um sintoma o qual não era apresentado antes, é liberdade tecnológica sendo utilizada de maneira correta.

1.1 OBJETIVOS

Neste capítulo, irei apresentar os objetivos da criação do aplicativo CUIDA, o qual tem como finalidade, ser uma via informativa sobre cuidados e precauções sobre sintomas e doenças.

1.1.1 OBJETIVO GERAL

O Objetivo Geral deste trabalho é apresentar o desenvolvimento de um aplicativo, para dispositivos móveis, usufruindo da linguagem Dart com o kit de desenvolvimento Flutter.

1.1.2 OBJETIVOS ESPECÍFICOS

Os Objetivos Específicos deste trabalho têm como foco principal:

- O agrupamento de informação sobre doenças a fim de alertar e educar sobre os processos preventivos dos mesmos.
- Tornar as informações, contidas no aplicativo, mais acessíveis aos portadores das doenças ou sintomas.
- Facilitar a busca por locais disponíveis para tratamento e consulta.

2 REFERENCIAL TEÓRICO

Por vezes, no levantamento de requisitos de um projeto, é questionado a tecnologia que será apresentada – além de ser a melhor que se adéqua – para sustentar a estrutura do projeto. Neste tópico, será apresentado as tecnologias para o desenvolvimento do aplicativo CUIDA.

2.1 Dart

Ao fazer o questionamento sobre qual tecnologia usar para determinado produto final é sem dúvida uma pergunta, que dependendo do sistema, pode ser difícil. E para este projeto foi pensado na estratégia de fazer uso de alguma linguagem que se encaixasse nos seguintes requisitos:

- Fortemente tipada;
- Cross Plataform.

Por conta desses fatores. Dart, desenvolvido pela Google, foi a melhor escolha para o projeto. A linguagem possui uma versatilidade para sistemas mobile, servidores e scripts de comando fáceis de serem implementados (GUEDES, 2018). Vendo isso, a biblioteca que usufrui da linguagem mais indicada para o projeto foi Flutter.

2.1 Flutter

Por ser uma biblioteca criada pela própria Google, o Flutter tem o foco na criação de aplicações móveis com Cross Plataform (BALTIERI, 2021). Isso quer dizer que ao gerar um aplicativo para celular com Flutter, ele terá a possibilidade de ser executado em Android e IOS, e com sua nova versão, a possibilidade para Web, é viável.

Um exemplo visual de como está a diferença entre Android e IOS no Flutter é com a imagem abaixo, que mostra uma tela do aplicativo nas duas estruturas:

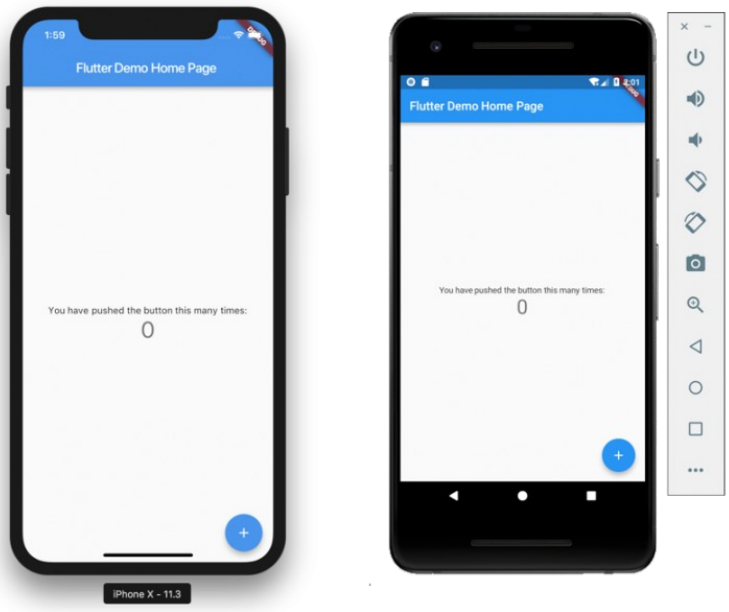


Figura 1 – Tela de aplicativo em Android e iOS Fonte: trustedreviews (2018a).

2.3 API

Application Programming Interface, mais conhecido como API, refere-se um conjunto de rotinas; criadas para interligar o banco de dados e sistemas com o cliente final, o qual utilizará alguma plataforma de comunicação (LIMA, 2019).

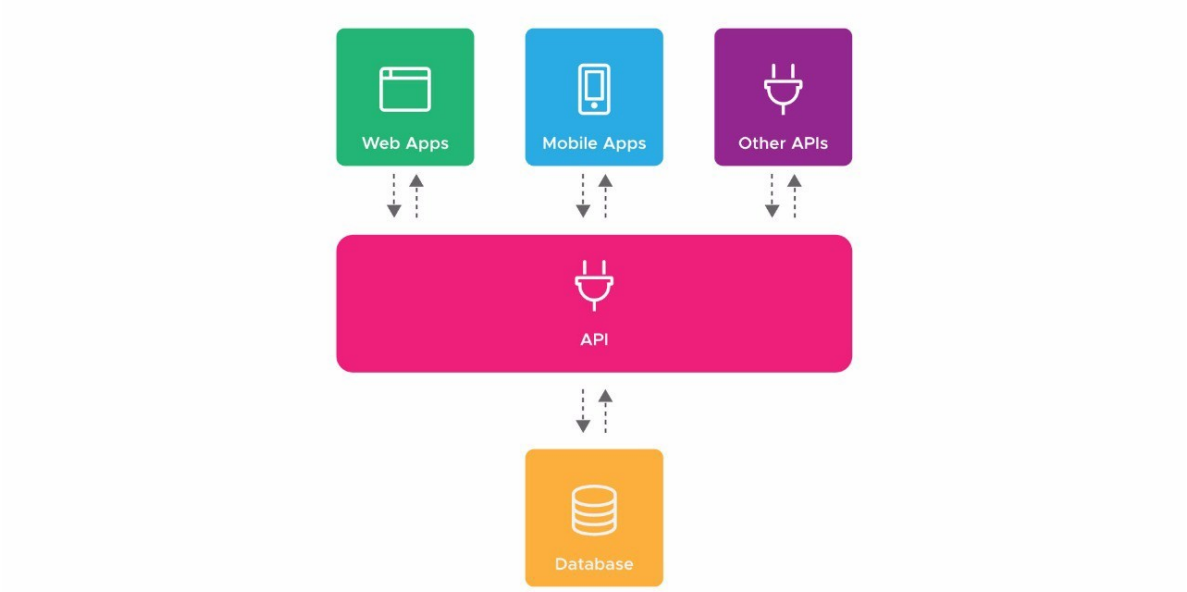


Figura 2 – Exemplo de funcionamento de uma API. Fonte: thiagolima_br (2019).

Uma API, de forma simplista, é um garçom. Quando você chegando a um restaurante, senta-se à mesa (frond-end) e faz um pedido, o garçom, ao anotar o seu pedido, leva até a cozinha (banco), fazendo assim a comunicação. A API é um meio, a forma mais estratégica de uma aplicação, podendo ser Web ou Mobile, de se comunicar com os dados presentes em um banco (REDHAT, 2017).

Neste projeto, utilizaremos duas APIs para fazer parte do sistema.

– Helth.Gov

Com o intuito de juntar esforços para prevenir doenças, o Departamento de Saúde e Serviços Humanos dos Estados Unidos, juntamente com o Gabinete do Secretário Adjunto para a Saúde, estão trabalhando para com outras empresas e agências da HHI (National Helth Iniciatives), *Iniciativa de Saúde Nacional,* fizeram uma API para informar sobre a prevenção de doenças, além de promover a saúde dos usuários finais com artigos e textos sobre determinados assuntos envoltos do mesmo.

Ela – A Helth.Gov – possui uma API aberta de consumo fácil e bem documentado. A API chamada de MyHelthfinder API está em sua terceira versão e possui *response* (Resposta a qualquer solicitação da API) em três maneiras distintas.

- JSON

JavaScript Object Notation, mais conhecido pelo acrônomo de JSON, é uma notação simples e leve que utiliza texto legível para transmitir os dados dos objetos.

- XML

Extensible Markup Language, utilizado para a criação de Notas fiscais, o XML também adota um padrão que pode ser utilizado para compartilhamento e transmissão de dados de uma API (CANALTI, 2017).

- HTML

No sistema da Helth.Gov o retorno da API também pode ser feito com uma página HTML. Este tipo de resposta, um HiperText, não será utilizada na aplicação, mas vale informar que um HTML nada mais é do que um bloco de tags.

– ApiMedic

Criado pela Priaid a ApiMedic é uma API médica com dados de doenças, sintomas, especialistas, recomendações e regiões do corpo – o qual referenciam os sintomas. Esta API possui duas vertentes: A paga e a gratuita. A paga está com o banco de dados mais atualizado possível sobre doenças e sintomas. A gratuita usufrui de um banco não atualizado, mas pode utilizar um número ilimitado de requisições.

2.4 SCRUM

Como forma de acelerar as entregas, produzir de forma mais organizada e assim consistir no fracionamento e constância de desenvolvimento inteligente, surgiu na década de 1990, os Métodos Ágeis.

Não distante de estruturas administrativas para gerar uma gestão de tempo. Os métodos ágeis surgiram para conduzir as equipes de desenvolvimento de software em processos que irão, ao final de tudo, fazer uma entrega mais satisfatória ao cliente.

Por ser uma das metodologias mais utilizadas da atualidade, *Scrum* usufrui da possibilidade de resolver problemas adaptativos e complexos, enquanto entrega produtos com maior valor possível de forma produtiva e criativa (FIA, 2017).

Sendo um Framework (estrutura genérica com o objetivo de prover uma função), o Scrum, é uma metodologia e possui um conjunto de boas práticas voltadas para o desenvolvimento de software. O Scrum pode ser utilizado para

outros setores fora o T.I, e por ser versátil desta maneira, ele foi adotado em empresas que fazem o bom uso desta metodologia.

Para este projeto, o Scrum fará parte do processo da gestão de micro entregas.

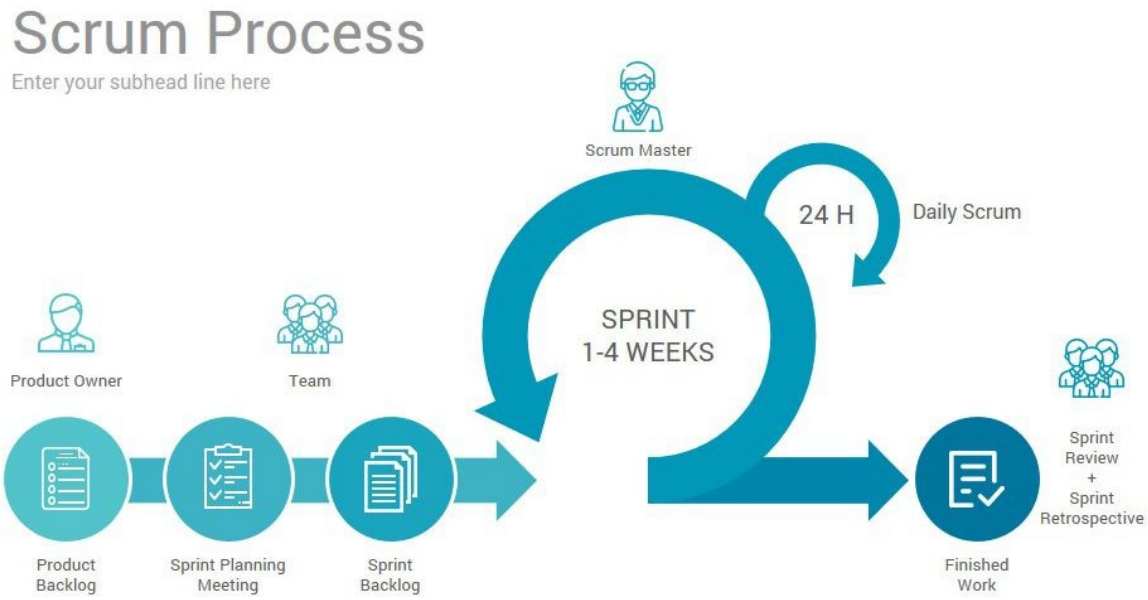


Figura 3 – Processo do Scrum. Fonte: blog.trainning.com.br (2020a)

Como forma de entender o Scrum, neste projeto iremos exemplificar os conceitos básicos do framework, a fim de facilitar no entendimento sobre o método ágil.

Chamadas de Scrum Team, as equipes de um projeto que utiliza Scrum, tem a divisão primária em três funções primordiais:

- Product Owner (PO) - Este é responsável pelo projeto. O PO se colocará como dono do projeto, a fim de que as entregas e o Product BackLog (Lista de Tarefas) sejam feitas.

- Scrum Master – Sendo a pessoa que irá definir o Backlog diário, o Scrum Master agirá como um líder de projeto com a finalidade de orientar os desenvolvedores ou equipe do projeto, a focarem em uma tarefa no ciclo de atividades, chamada de Sprint.

- Scrum Team – Consiste na equipe que irá desenvolver o projeto. De acordo com as informações do Scrum. Cada pessoa dentro do Scrum Team pode definir as

atividades a serem focadas no ciclo de atividades diárias. Com as reuniões diárias, chamadas de *Daily Scrum,* o Scrum Master poderá gerenciar os ciclos de entrega, fazendo assim, as micro entregas para o PO.

O desenvolvimento do CUIDA foi baseando-se no modelo do Scrum. Por conta da equipe ser formada apenas por mim, foi criado um o Product Backlog e o Sprint Backlog com a finalidade de organizar as atividades prescritas em um tempo hábil. Prosseguindo com isso, o desenvolvimento do sistema foi feito em sprints após o escopo do projeto ter sido bem definido.

3 METODOLOGIA

A fim de realizar este projeto, a pesquisa foi classificada como exploratória devido ao fato de esta, ser a criação e o desenvolvimento de um aplicativo baseado nas necessidades e remoção de dúvidas de pacientes sobre questões de saúde. Além disso, foi necessário para o desenvolvimento deste projeto, a aplicação de multi-métodos, levantados em duas etapas:

1 – **Conceitos teóricos** – Aqui é analisado a estrutura utilizada, conceitos básicos que serão necessários para a criação do aplicativo, bem como o desenvolvimento do projeto.

2 – **Pesquisa, desenvolvimento e testes** – Nesta etapa uma pesquisa pelas necessidades referente ao projeto será levantada, bem como, a produção do aplicativo baseado nas telas desenhadas e os requisitos levantados. Nesta etapa de produção, o aplicativo CUIDA será produzido em duas etapas:

a. - Escolha de Back-end.

A escolha de uma API que sirva como back-end para este projeto e atenda às necessidades da implementação das telas a serem produzidas, faz-se necessário para que as consultas de dados sejam feitas. Por meio disso, o aplicativo CUIDA vai usufruir de duas APIs:

- Helth.gov – Do governo dos Estados Unidos da América. Uma API que gerará informação - Artigos e textos – sobre temas diversos que englobam saúde e bem-estar.

- ApiMedic – Uma API de consulta baseada nos sintomas que a pessoa está sentindo no momento. Esta possui uma vasta biblioteca de informação sobre estado do sintoma, perguntas relacionadas ao sintoma, locais do corpo, direcionamento médico - informando qual médico adequado a aquela doença possível - e diagnóstico.

b. - Desenvolvimento do Front-end.

A utilização do Flutter o qual foi desenvolvido pela Google, oferece suporte para a estrutura desenhada do aplicativo CUIDA. Flutter possui bibliotecas e componentes externos que serão utilizados no projeto. O *cupertino_icons,* uma biblioteca de ícones que será usada como padrão do projeto. O *dio,* uma biblioteca de comunicação com API, entre outros que serão apresentados no decorrer do projeto.

4 DESENVOLVIMENTO

Neste capítulo irei descrever os tópicos e estruturas criadas, bem como a utilização dos métodos propostos e a tecnologia usada. Será feito da mesma forma, a descrição do que foi criado em cada Sprint.

4.1 Flutter

Na criação desde projeto, a utilização da linguagem de programação Dart e o framework Flutter fizeram a diferença para a facilidade do entendimento, bem como a separação de atividades diante dos requisitos levantados no Backlog.

Uma diferença do Flutter para as demais linguagens é a sua alta performance e a característica de basear sua construção em *widgets*.

Em Flutter os widgets são partes de bibliotecas ou componentes nativos do próprio Flutter que vão, de forma abstrata, se empilhando para formar os componentes que irão ser apresentados em tela. Por conta disso, a possibilidade de construção dos aplicativos de forma mais performática possível é viável, além da capacidade de atuação da linguagem como um sistema nativo, mesmo sendo híbrido.

Por conta da construção da interface ser baseada em blocos chamados de widgets, estes são agrupados sobre dois tipos principais de widgets que estruturarão uma tela no Flutter: O Stateful e o Stateless.

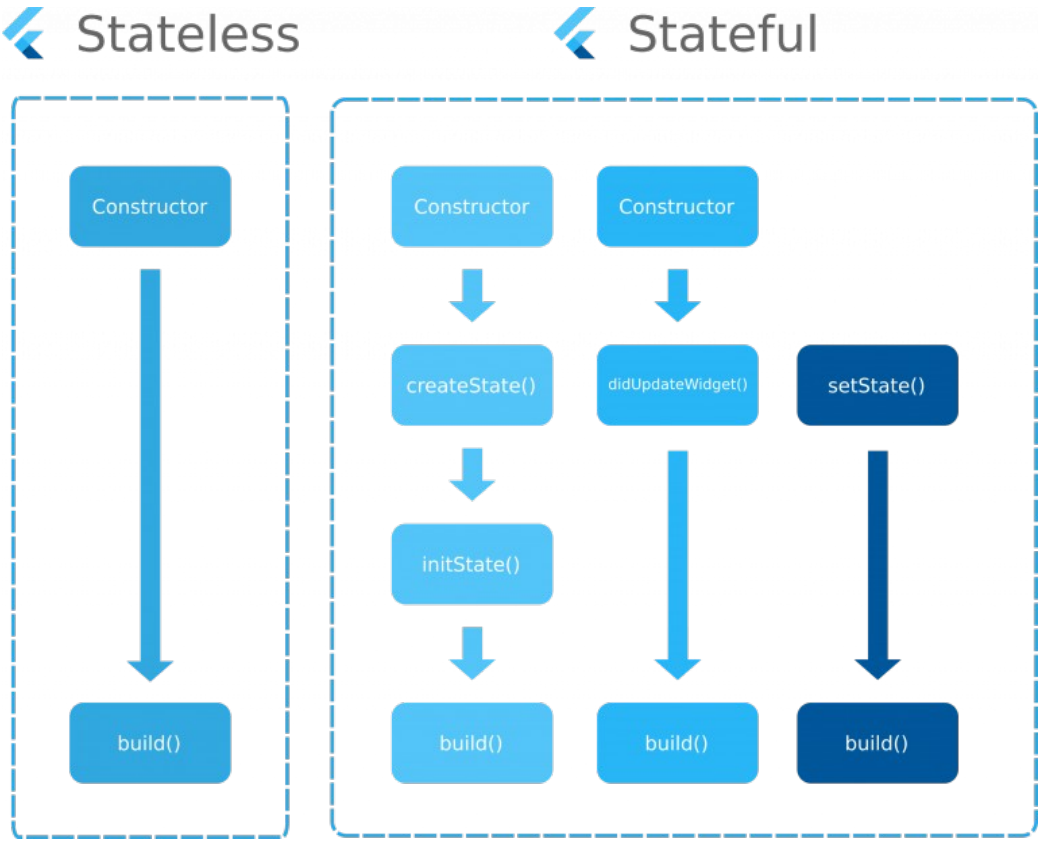


Figura 4 – Tipos de Widgets de Estado no Flutter. Fonte: Flutterclutter.dev (2020a).

Como mostrado na figura acima as duas principais classes no Flutter fazem a gerência de estado. Quando uma tela é criada, ou até mesmo um *widget*, os dados são previamente carregados, caso este às possua. Após o carregamento da tela, caso o *widget* necessite sofrer alteração do estado em que se encontra, o ideal é que ele seja criado como Stateful, pois aquele *widget* em específico sofrerá alteração de estado, enquanto uma página Stateless, não. (MARC, 2020).

```dart
class ExternalStateOnlyWidget extends StatelessWidget {
  ExternalStateOnlyWidget({
    @required this.textString,
    @required this.changeText
  });

  final String textString;
  final Function changeText;

  @override
  Widget build(BuildContext context) {
    @override
    Widget build(BuildContext context) {
      return ListView(
        children: <Widget>[
          RaisedButton(
            child: Text('Press'),
            onPressed: changeText,
          ),
          Text(textString)
        ],
      );
    }
  }
}
```

Figura 5 – Exemplo de Classe Stateless. Fonte: Flutterclutter.dev/ (2020a)

Um bom exemplo de classe Stateless é uma tela de login. Os dados são preenchidos no campo de e-mail e senha e ao apertar no botão, é feita a comunicação até a API e assim é verificado se os dados são válidos ou não. Esta tela, por apenas fazer a verificação de dados, não precisar de um recarregamento e não possuir variáveis que ter terão seus dados alterados, não necessita ser uma página Stateful.

```dart
class InternalAndExternalStateWidget extends StatefulWidget {
  InternalAndExternalStateWidget({
    @required this.buttonText,
  });

  final String buttonText;

  @override
  State<StatefulWidget> createState() {
    return _InternalAndExternalStateWidgetState();
  }
}

class _InternalAndExternalStateWidgetState extends State<InternalAndExternalStateWidget> {
  String text = 'Initial';

  @override
  Widget build(BuildContext context) {
    return ListView(
      children: <Widget>[
        RaisedButton(
          child: Text(widget.buttonText),
          onPressed: _changeText,
        ),
        Text(text)
      ],
    );
  }

  void _changeText() {
    setState(() {
      text = 'Overridden';
    });
  }
}
```

Figura 6 – Exemplo de Classe Stateful. Fonte: Flutterclutter.dev/ (2020a)

Por ser uma *widget* de gerência de estado. Uma página ou widget Stateful utilizará de mais recursos que uma Stateless, sendo assim ela será mais pesada no carregamento e consequentemente fará mais uso da memória do celular para fazer a gestão dos estados dos dados a serem carregados.

Ao ser feito o levantamento e o design de telas do sistema, a decisão sobre qual tipo de *widget* utilizar em cada componente ou tela será essencial para a gerência do celular e do desempenho para a execução do aplicativo. Ao utilizar Stateful quando apenas Stateless seria necessário, seria como estar tentando matar uma mosca com uma bazuca.

4.2 SPRINTS

A ser explicado como a metodologia ágil Scrum funciona, no item *4.2,* irei exemplificar como cada *sprint* se deu até o iniciar do projeto levando consigo as primeiras requisições. Cara *sprint* possui uma listagem de tarefas a serem finalizadas em determinado tempo. Assim, o aplicativo CUIDA precisou de levantamentos concisos distintos, devido fatores como: pessoa única no projeto, tempo hábil para desenvolvimento e planejamento.

4.2.1 SPRINT 01 – Escopo do Projeto, Levantamento de Requisitos e Problemática.

4.2.1.1 - Planejamento

A importância de um primeiro Sprint é grande ao ponto de impactar no restante do projeto. Podendo acarretar, caso mal planejado, re-fatoração de código - nos melhores casos -, quebra de escopo de projeto, perca de tempo em código que não será utilizado, inadequação de equipe – especialistas em linguagens que não serão necessárias -, perca de credibilidade com os *stackholders* (partes interessadas), etc.

Visando isso, o aplicativo CUIDA teve uma parcela grande de seu projeto voltado para a primeira Sprint, Cuja mesma foi feita a realização do levantamento

inicial da problemática: “Como facilitar o acesso a informação de doenças para a pessoa comum”.

Inicialmente o projeto CUIDA, no primeiro levantamento e reunião com o professor responsável por este trabalho, estava voltado para a criação de um aplicativo com o seguinte enfoco:

- Ser um aplicativo de informação médica voltada para pacientes com câncer.

- Ele teria um foco mais humanitário na tratativa com o usuário, com o intuito de informar artigos sobre cuidados pessoais e motivacionais.

- O ponto principal do CUIDA seria a informação sobre sintomas, e este ainda é. Porém, nesta versão de planejamento o aplicativo voltaria apenas para sintomas de câncer em específico: sintomas do câncer de pele, mama, etc.

- O aplicativo teria geolocalização com o intuito de mostrar clínicas, postos de saúde, locais de terapia e grupo de apoio para pessoas com câncer.

- Como a temática de câncer voltada para o projeto, foi reparado uma falta de informação maior para o grupo masculino, propagandas e incentivos para o teste do câncer de mama em mulheres é maior, mas a temática do câncer de próstata ainda é tratada com certa vergonha até pelo próprio público. Devido a isso, a versão inicial de lançamento do projeto iria ter a primeira temática voltada para os homens.

- Geração de Perfil, uma página voltada o perfil do usuário acompanhar o seu progresso bem como visualizar mensagens automáticas para o usuário de acordo com o perfil.

Esses foram os pontos iniciais levantados no primeiro ciclo. O enfoco deste, no desenvolvimento, foi a estruturação inicial do projeto.

- Estruturação inicial do projeto.

- Fazendo a criação do projeto.

- Iniciando o repositório no GitHub.

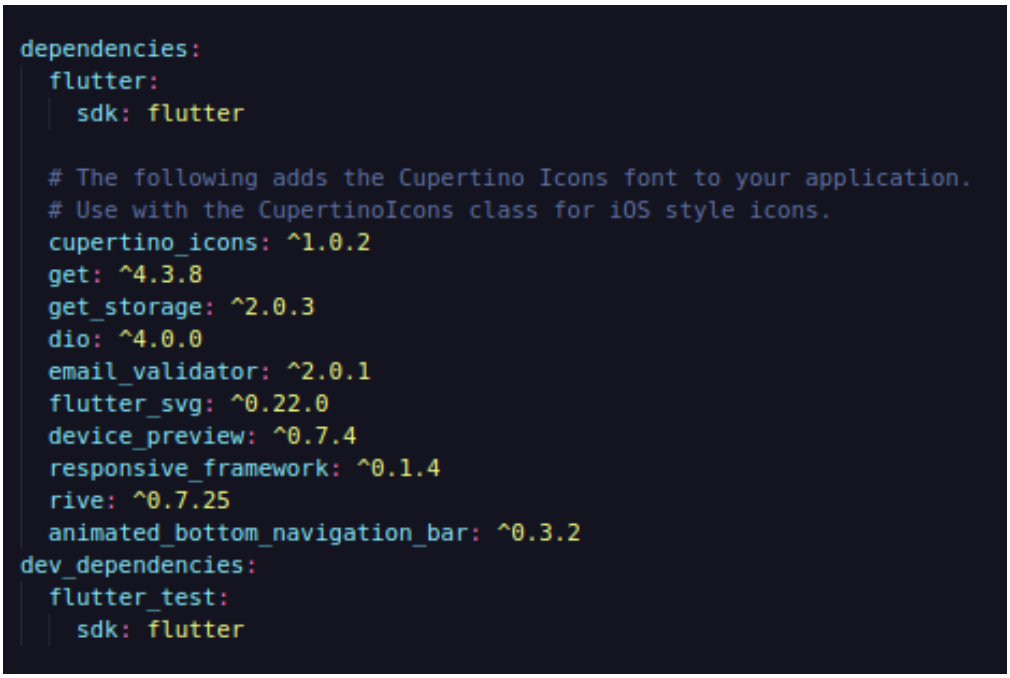
- Levantamento de bibliotecas a serem utilizadas.

Os quatro requisitos para este primeiro ciclo, além da estruturação do projeto, fazem parte da criação da base de um projeto em Flutter. Também foi adicionado o último item referente as bibliotecas utilizadas.

4.2.1.2 - Desenvolvimento

O cerne do desenvolvimento do projeto neste primeiro sprint, foi a estruturação do aplicativo. Logo, para isso fez-se criação de um projeto inicial na máquina utilizando o terminal do Linux (Ubuntu 20.04). Primeiramente é necessário a instalação do Flutter SDK (*Software Development Kit*)*, o Kit de Desenvolvimento de Software Flutter.* Para isso, no terminal foi utilizado a linha de comando: $ Flutter sdk-path. Este comando instala toda a biblioteca do Flutter necessária para o projeto. Foi instalado o Android SDK para utilização de emuladores e bibliotecas do Google, e por fim foi utilizado o comando: *$ flutter create tcc_projetc* o qual criou uma pasta com os arquivos padrões de um projeto em Flutter.

Diante da base criada foi excluído arquivos desnecessários e instalado as bibliotecas padrões utilizadas em projetos como este.



```
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.2
  get: ^4.3.8
  get_storage: ^2.0.3
  dio: ^4.0.0
  email_validator: ^2.0.1
  flutter_svg: ^0.22.0
  device_preview: ^0.7.4
  responsive_framework: ^0.1.4
  rive: ^0.7.25
  animated_bottom_navigation_bar: ^0.3.2
dev_dependencies:
  flutter_test:
    sdk: flutter
```

Figura 7 – Dependências do Projeto CUIDA Fonte: Elaborado pelo Autor (2021a)

A biblioteca *cupertion_icons* vem como padrão para a utilização do projeto em IOS, O *get*, é uma biblioteca de maior uso dentro da comunidade, pois ela faz a gerência de estado, estado de navegação, gerência de dependência e comunicação com API. O *get_storage* é uma sub biblioteca do *get* que faz a gerência do estado dos itens em memória.

O *dio* é uma biblioteca de comunicação com API, esta é a que será utilizada para fazer a comunicação com a API no sistema do CUIDA, pois mesmo que a

biblioteca do *get* faça isso, o *dio* consegue oferecer um serviço de melhor qualidade, devido a este ser seu único foco.

Temos o *email_valitator*, uma biblioteca que validará o e-mail do usuário, este ponto será implementado numa versão futura do CUIDA o qual será falado mais a frente. Para imagens em svg como a splash art temos o Flutter_svg que será usado no próximo *sprint*. Para emular dentro de um emulador, um IOS ou Windows Phone, o *device_preview* irá gerar uma tela similar de acordo com o modelo desejado, útil para a responsividade.

Ainda dentro do tema de responsividade e tela, temos o *responsive_framework*, uma biblioteca que fará uso muito parecido de escalas do Bootstrap (biblioteca de CSS para web) para itens dentro do projeto. Será também usado no próximo *sprint* o *rive*, uma biblioteca interativa de design e animação para ler determinados arquivos de animação, muito usado para splash arts e gits dentro do projeto.

E por fim o *animated_bottom_navigattion_bar*, uma biblioteca que trará uma barra de navegação customizável, útil para projetos em que a barra de navegação foge dos padrões.

A fim de salvar o projeto e ainda assim gerenciar o estado dele, foi criado um repositório privado no GitHub com os dados, informações do projeto e listagem de atividades seguintes.

4.2.2 SPRINT 02 – Criação da Splash Art, Gestão de Rotas e Busca de Assets.

4.2.2.1 - Planejamento

Com o desenvolvimento do primeiro *sprint* feito, o salvamento das atividades feita o projeto sendo feito de forma quase que diária, o segundo sprint se baseou no processo de criação da parte inicial do aplicativo CUIDA:

- Criar uma SpashArt o qual carregará a logo com uma temática do aplicativo.
- Redirecionamento de Tela para Login ou similar com Routes.

A criação da tela de Splash Art foi baseada em um aplicativo que desenvolvida junto da equipe do trabalho. Nela, me baseei em uma estrutura que utilizasse o sistema Rive. Este por sua vez é uma plataforma de criação, design e

animação online, um estúdio para artistas criarem gifs ou curtas. Nesta mesma plataforma, ela pode ser externalizada para a utilização de sistemas, o qual designers produzem o conteúdo no Rive e disponibilizam em um arquivo de extensão *riv*.

Figura 8 – Tela de SpashArt Aplicativo CUIDA. Fonte: Elaborado pelo Autor (2021a).

4.2.2.2 - Desenvolvimento

Com a animação feita, seria necessário fazer a criação das rotas e direcionamento das telas de acordo com o levantamento preestabelecido. Dentro de um projeto de Flutter o arquivo principal leva o nome de Main.dart, este por sua vez faz o direcionamento ou até mesmo o carregamento do projeto.

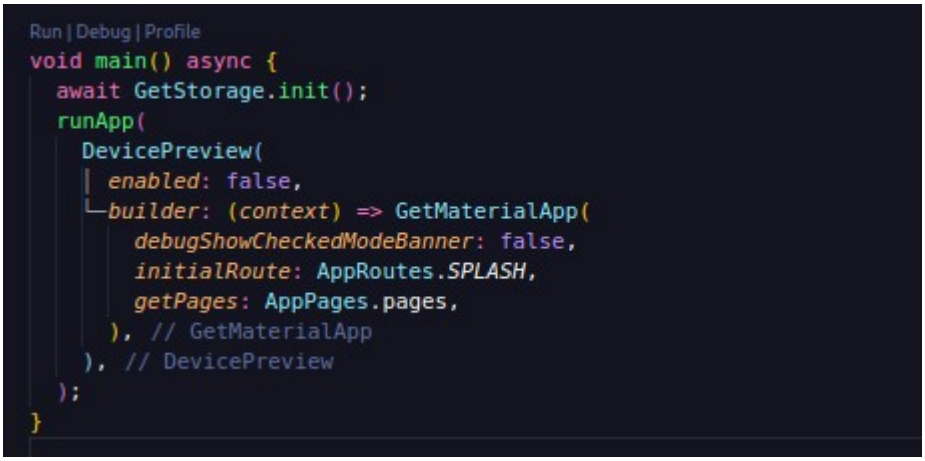


Figura 9 – Classe Main.dart – Cuida. Fonte: Elaborado pelo Autor (2021a)

Como observado na imagem, a biblioteca do DevicePreview que emula telas dentro de um emulador do android está desativada no momento, e o build, que é a geração do aplicativo parte para uma classe *widget* chamada de GetMaterialApp que faz parte da gerência de rotas do *get*.

Sabendo disso para iniciar a fase de rotas em initialRoute é necessário criar uma classe que irá gerenciar as rotas das páginas dentro do projeto. Neste caso em específico foi criado a AppPages e a AppRoutes.

Figura 10 – Classe AppPages – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Como todo projeto que possua uma estrutura de *frontend* bem desenhada. Um arquivo de rotas é necessário para direcionar as páginas e assim modelar a disposição das telas do projeto. No aplicativo CUIDA a gerência de rotas fica a cargo da AppPages que dispõe de uma listagem estática de páginas que dependem da biblioteca do get, o GetPage. Nela é definida o nome da página, que estende de AppRoutes; uma *page* que é tomada por uma arrow function que chama a página em questão e um binding, que no Flutter é responsável por indicar o carregamento necessário de controllers e dependências antes do carregamento da página em si.

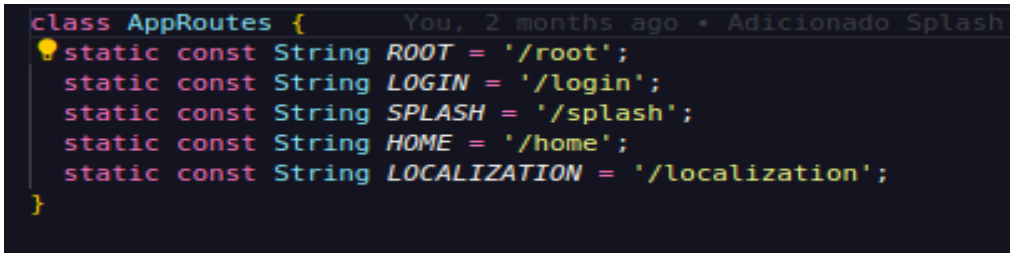


Figura 11 – Classe AppRoutes – CUIDA. Fonte: Elaborado pelo Autor (2021a)

4.2.3 SPRINT 03 – Modificação do Escopo, Barra de Menu e HomePage

4.2.3.1 - Planejamento

Com o arquivo de rotas bem desenhado, o redirecionamento de tela após a tela de SpashArt deveria ir para o uma tela de login, como todo aplicativo que utiliza autenticação de usuário para usar o sistema. Entretanto, nesta primeira versão do CUIDA, os usuários não teriam necessidade de se autenticar já que as APIs propostas seriam apenas para consulta, não havendo necessidade de armazenamento de dados dos usuários.

Porém, baseado nos critérios de uma reunião que obtive com o professor responsável levantamos a seguinte questão: Achar uma API feita que alimentasse o sistema, baseado apenas em dados referente a câncer.

Não seria uma tarefa simples e nas pesquisas levantadas, nenhuma API de consumo gratuito ofereceria algo tão específico quanto câncer. Devido a tal fator, e principalmente ao tempo de criação do projeto, o aplicativo necessitava de uma solução. Devido a isso, a especificidade que seria o câncer foi alterada para doenças em geral. A partir do *sprint* desta reunião, o aplicativo CUIDA iria focar nas tratativas humanitárias, geolocalização, artigos e agora, doenças de forma ampla.

4.2.3.2 - Desenvolvimento

Como parte do desenvolvimento deste ciclo, o direcionamento das rotas concluído abriu a possibilidade de uma gerência melhor da organização e distribuição das telas.

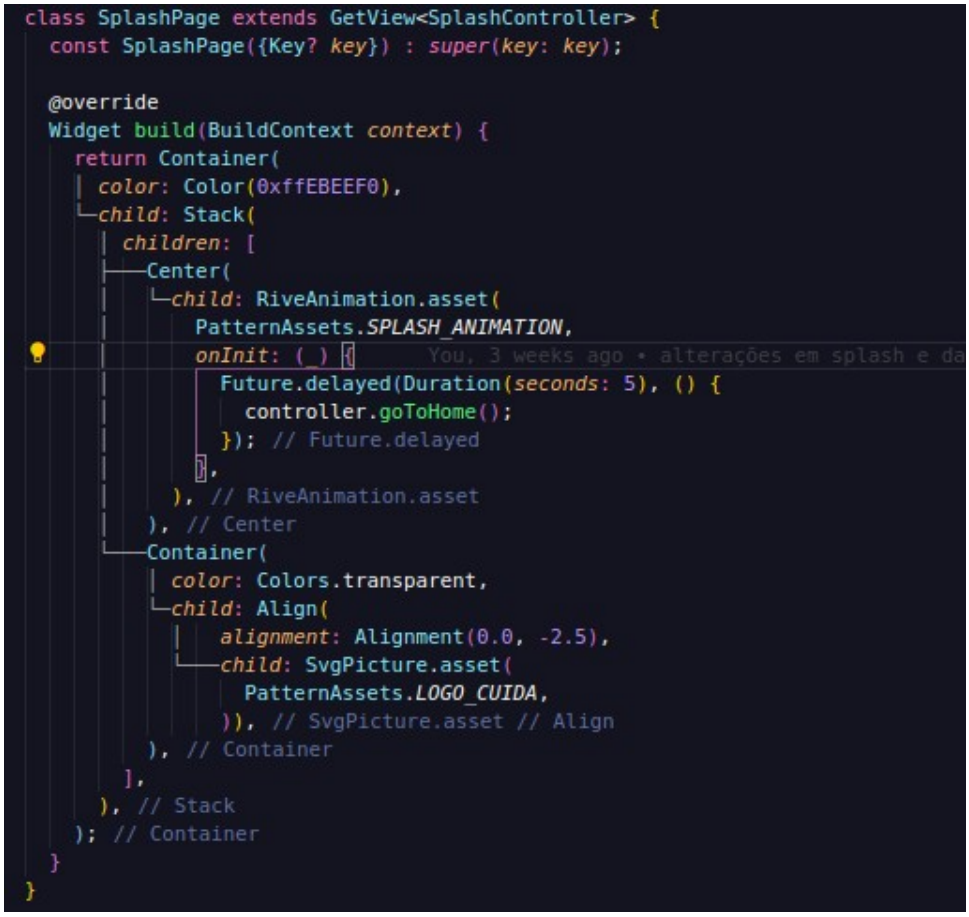


Figura 12 – Classe SpashArt – CUIDA. Fonte: Elaborado pelo Autor (2021a)

A Classe da SplashArt, nomeada de SpashPage, estende da SplashController o qual terá as funções e métodos que irão fazer a comunicação necessária. Antes deste método, é possível visualizar um *widget* chamado de *Future.delayed* o qual aguardará a chamada do método por um tempo de duração de cinco segundos. Nesta tela também, percebe-se o carregamento de um método chamado de goToHome, que utilizando a biblioteca do *get*, chamará a seguinte tela que será, neste caso, a tela de RootPage.

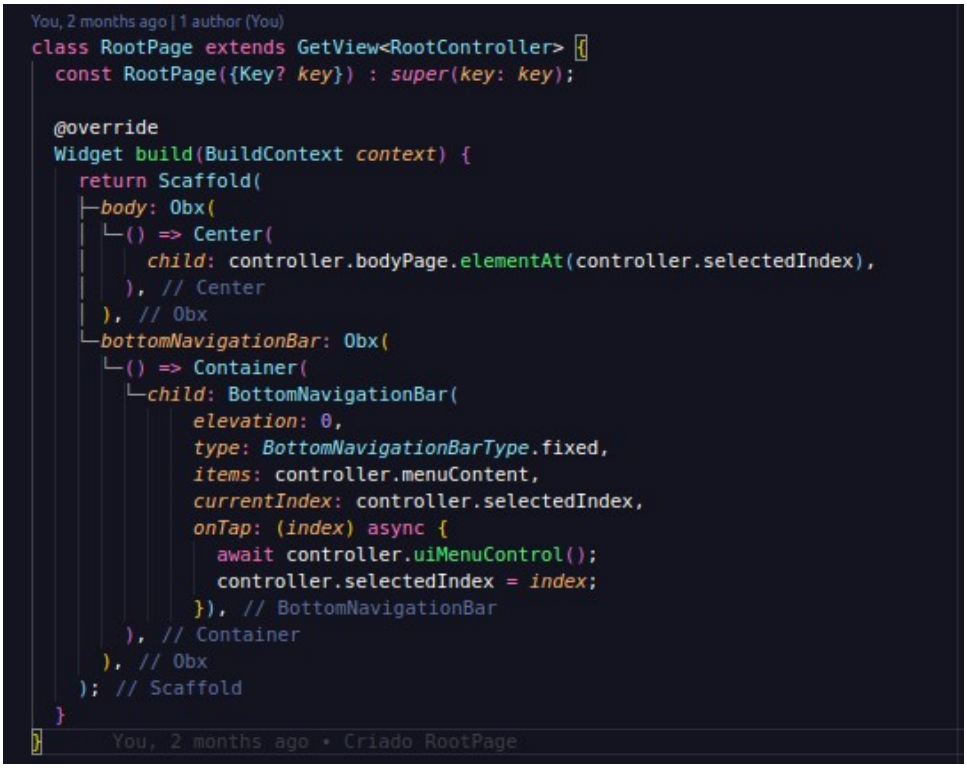


Figura 13 – Classe RootPage – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Neste ponto em específico do projeto a RootPage funcionará como um *SPA (Single Page Application).* Nela, as telas irão se alternar no elemento *body* do *widget* Scaffold, como mostrado na figura acima. Nesta mesma página, foi implementada também a barra de menu no elemento bottomNavigationBar. Como mostrado na imagem, tanto no *body* quando no bottomNavitagionBar possuem um *widget* em específico chamado de Obx

Como falado no tópico 5.2 sobre Flutter, existem dois tipos principais de *widgets* o Stateless e o Stateful. O primeiro, não trabalha com a gerência de estado e por conta disso torna-se mais leve, o mais adequado para telas que não utilizam objetos que alteram seus dados. Contudo, com a biblioteca do get, foi criada a possibilidade de fazer a alteração de estado de um objeto sem a necessidade de usar uma extensão da classe Stateful. O nome deste widget responsável para gerir o estado e as alterações vindas de uma resposta na controller chama-se Obx.

Com o Obx o *widget* subsequente torna-se ‘ouvido’ e qualquer alteração em seu estado reflete na tela o qual ele está sendo mostrado. Normalmente estes objetos são retornos dos estados das variáveis declaradas na controller da página em questão.

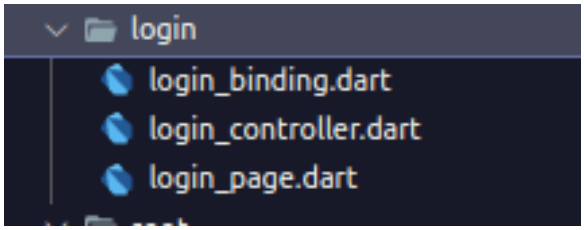


Figura 14 – Exemplo de organização de Pastas – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Devido ao estado sendo controlado na controller toda e qualquer comunicação vindo dela (controller) irá refletir nos dados e informações na tela em questão. Como no exemplo acima, a tela de login o qual não foi implementada nesta versão do projeto (está apenas comentada), inicia da seguinte maneira:

- O arquivo de binding inicia informando quais controllers, repositórios ou interfaces a controller da tela irá necessitar.

- A controller em questão chamará os métodos necessários.

- E por fim será carregada a tela, chamada de *page*.

Este é um ciclo normal de funcionamento no Flutter. A estrutura base para o carregamento de telas e comunicação dependerá do projeto, mas a organização básica irá mensurar o consumo de memória, e levando isso em consideração, as páginas deste projeto estenderão apenas de classes Stateless utilizando a gerência de estado do get (Obx).

A partir da chamada na controller com os métodos sendo carregados, os dados aparecem em tela. Na imagem abaixo pode-se ter uma noção de como funciona a chamada dos métodos:

```
void onInit() {
  mountMenu();
  super.onInit();
}

Future mountMenu() async {
  if (1 == 1) {
    menuContent.value = [
      BottomNavigationBarItem(
        icon: Icon(CupertinoIcons.home),
        label: "Ínicio",
      ), // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Icon(CupertinoIcons.doc_chart),
        label: "Acompanhar",
      ), // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Icon(CupertinoIcons.location),
        label: "Local",
      ), // BottomNavigationBarItem
      BottomNavigationBarItem(
        icon: Icon(CupertinoIcons.person),
        label: "Perfil",
      ), // BottomNavigationBarItem
    ];
    return bodyPage.value = [
      HomePage(),
      LoginPage(),
      LocalizationPage(),
      // ProfilePage(),
    ];
  }
}
```

Figura 15 – Método de Montagem do Menu – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Dentro de uma controller existem vários métodos que por padrão auxiliam na programação em Flutter e o onInit() é uma delas. Por padrão ela chamará uma função antes do carregamento de tela. E neste caso em específico está chamando de forma ainda não assíncrona, o mountMenu(). Nela, o qual é responsável pelo carregamento de dados no menu, pode-se perceber que existe um *async* indicando que a função é assíncrona. Dentro dela, em seu escopo há uma condicional previamente setada, esta irá servir para construção das futuras condicionais o qual irão mostrar menus de acordo com o tipo de usuário.

Por fim, é implementado dentro de um array o carregamento das páginas, o qual ao serem pressionadas por um botão com ícone, irão direcionar para a tela clicada. Mas antes disso, a tela de HomePage precisa ser criada.

4.2.4 SPRINT 04 – Telas Principais, pesquisa API.

4.2.4.1 - Planejamento

Por conseguinte, a criação das gerências de estado na RootPage e a barra de menu, o próximo passo será:

- Desenhar o layout das telas principais.
- Procurar as APIs que farão parte do projeto.
- Rotas de telas subsequentes.
- Utilidades

A partir daqui, segue-se implementando de acordo com a pesquisa referente a API. O último tópico (Utilidades), refere-se a uma pasta chamada de *utils* dentro do sistema, o qual irá usufruir de classes padrões para projetos, já pré-estabelecidos, por exemplo:

- Máscara de CPF/CNPJ
- SnackBar
- Classe de Conexão/*Repository*

A partir das funcionalidades que forem sendo necessárias, será implementada nesta pasta.

4.2.4.2 - Desenvolvimento

Para este ciclo foi necessário efetuar pesquisas de layout para estruturar uma base, um padrão que precisa ser seguido ao desenvolver um sistema no *frontend*. E para isso o layout foi baseado nas seguintes telas:

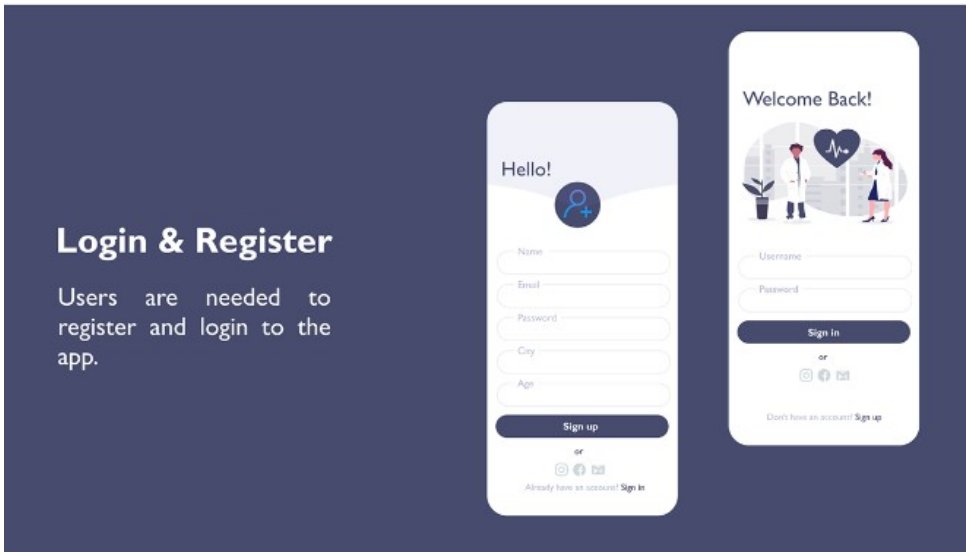


Figura 15 – Tela de Login MedicSystem. Fonte: Behance (2021a)

Figura 16 – Telas do aplicativo MediCare. Fonte: Behance (2021a)

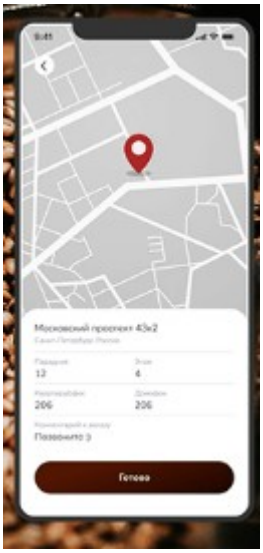


Figura 17 – Telas do aplicativo CoffeeShop. Fonte: Behance (2021a)

Utilizando de padrões presentes nas telas, pode-se desenhar a tela de HomePage, que informará artigos bem como, dará boas-vindas ao usuário.

Contudo, a principal fonte de dados ainda estava sendo pesquisada entre este *sprint* e o anterior. E para conseguir dar aos usuários os artigos necessários, foi feito pesquisas relacionadas a consumo de informação, doenças e pesquisas acadêmicas.

Durante o processo a melhor API que, de forma gratuita, consegue expor melhor as condições informativas que a aplicação necessitava, foi a MyHelthfinder, do Governo dos Estados Unidos da América.

Nela, localizada no site *helth.gov*, possui documentação acessível, endpoints em três tipos de retornos bem como no idioma inglês e espanhol.

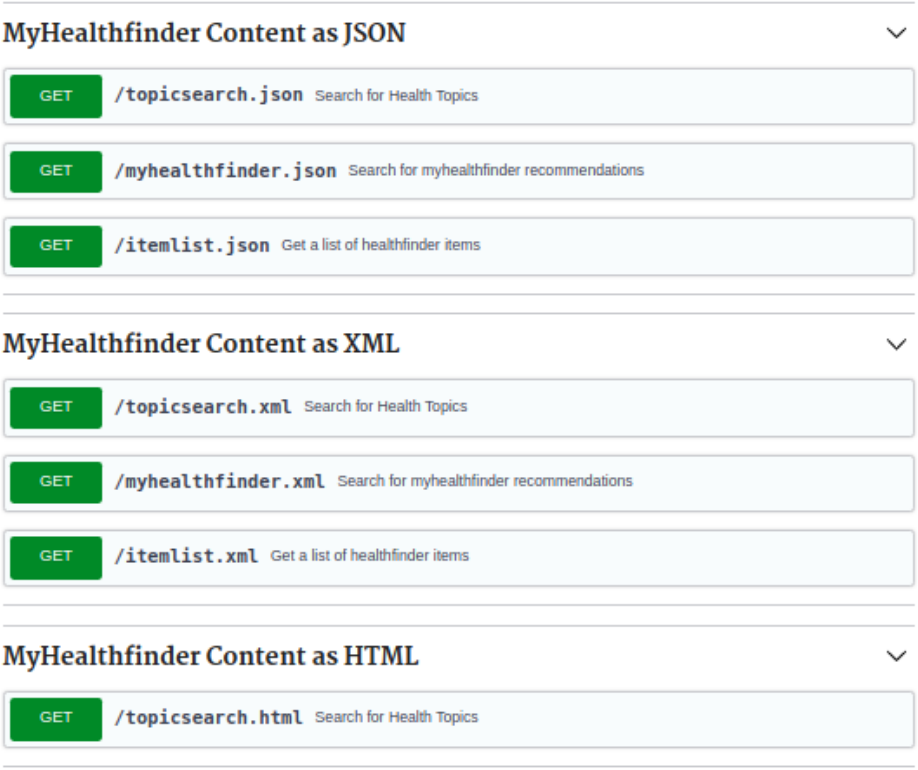


Figura 18 – EndPoints MyHealthfinder. Fonte: Health.gov (2021a)

Por estar utilizando a biblioteca get, a preferência pelo retorno foi em JSON, outro ponto também é a praticidade que este tipo de retorno oferece: facilidade na leitura. Dentro da resposta da API, após informar o conteúdo no corpo da requisição, diversos tópicos são informados, tais como: sexo, idade, se está gestante ou não, palavras-chave, sexualidade ativa, entre outros; todos estes tópicos estão dentro de um *array* chamado *de query.*

Seguidamente, ainda dentro do JSON de retorno, temos a parte principal do consumo desta API que é o *array* chamado Resources. Nele é onde o conteúdo é mapeado para a utilização do sistema CUIDA.

Vários tópicos, artigos e informativos sobre saúde em geral vem no retorno da consulta.

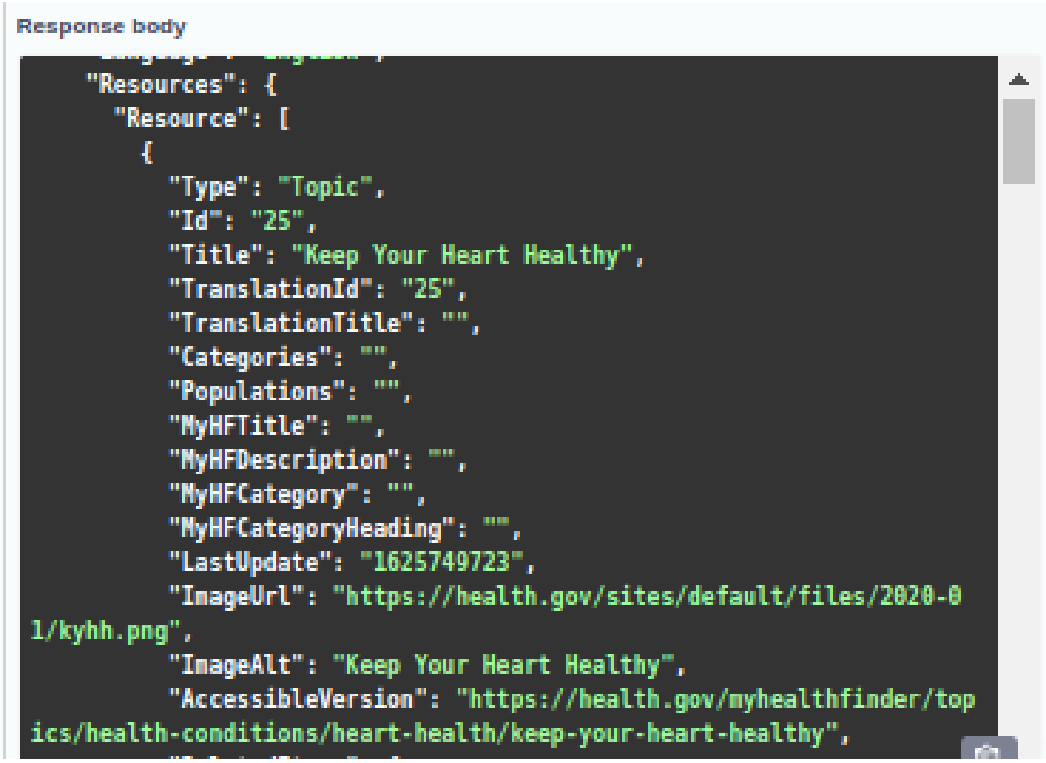


Figura 19 – Resources List - MyHealthfinder. Fonte: Health.gov (2021a)

Dentro de Resources, a listagem de artigos se subdivide em tópicos. Como na imagem acima, o primeiro tópico em *title* refere-se a “Mantenha seu coração saudável” (Keep Your Heart Helthy). Ainda neste tópico terá mais uma listagem de conteúdos dentro de Section.

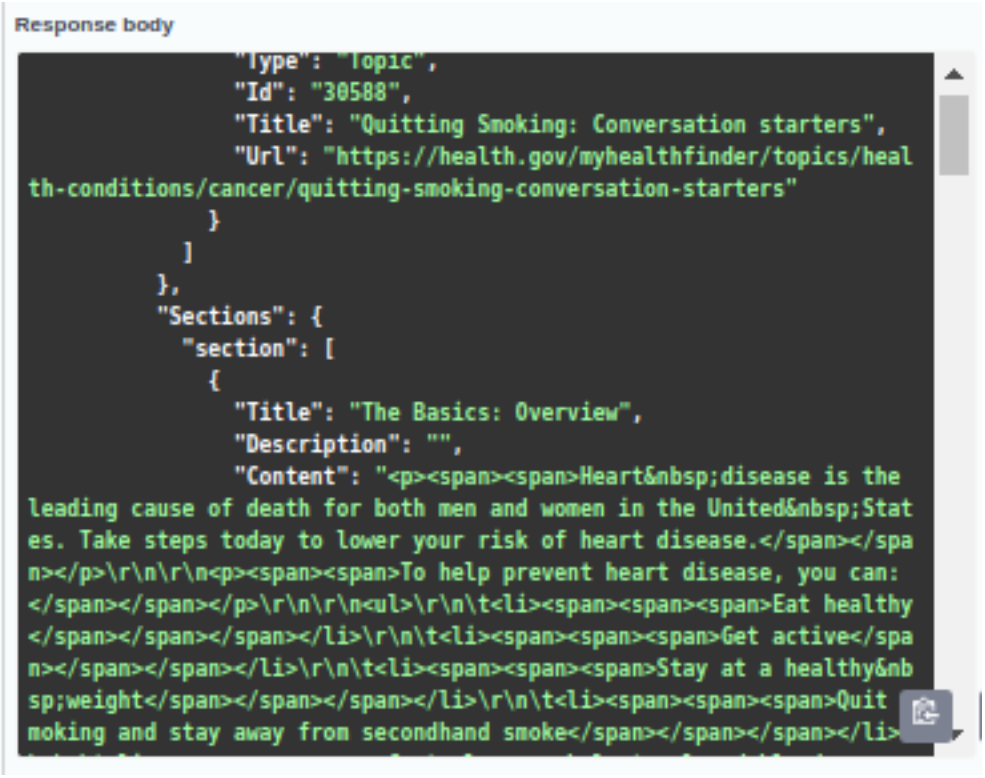


Figura 20 – Section List - MyHealthfinder. Fonte: Health.gov (2021a)

A partir do campo Sections, a listagem dos artigos continua, desta vez, está relacionada ao tópico principal. Dentro dele (*array* de *section*) separado em título, descrição e conteúdo que, como valor, está sendo retornado em HTML.

A API seguinte englobará a pesquisa de doenças. Este ponto possui um peso importante, pois ele é a parte principal do projeto:

- Facilitar o acesso à informação sobre doenças em geral para os usuários.

Com a busca para os artigos finalizada e a modelagem para a busca de artigos sendo preparada para a *sprint* seguinte, ao passo que as informações colhidas serão entregues na página inicial e, seguidamente ao clicar em um tópico da listagem, abrir uma página com detalhes do artigo, as rotas destas páginas está alinhada com os próximos passos do projeto.

Contudo, para a próxima API foi-se necessária uma pesquisa com a finalidade de que o retorno fosse claro e objetivo o suficiente, bem como os detalhes da doença ou sintoma. Ocorreu de haver APIs gratuitas com tempo de consumo limitado, APIs pagas que informavam dados inválidos ou desatualizados, mas nada se sobrescreveu tanto as necessidades quanto a APIMedic.

Esta é uma API que possui diversos idiomas de consumo (excluindo português) e duas versões:

- Uma paga, o qual você tem os dados mais atualizados com número limitado de consultas.
- E uma gratuita o qual permitirá fazer múltiplas consultas, porém com dados não tão atualizados.

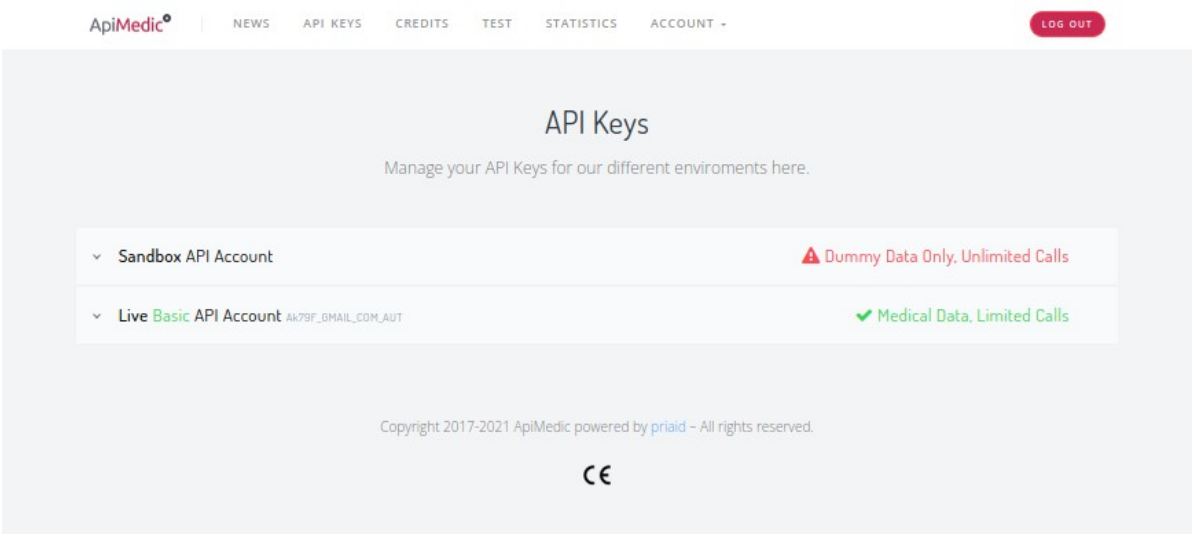


Figura 21 – APIs - APIMedic. Fonte: apimedic.com (2021a)

A APIMedic possui endpoints com várias finalidades:

- Busca por sintomas
- Diagnótisco baseado nos sintomas e localização do corpo.
- Especialistas recomendado, baseado no sintoma e diagnóstico.

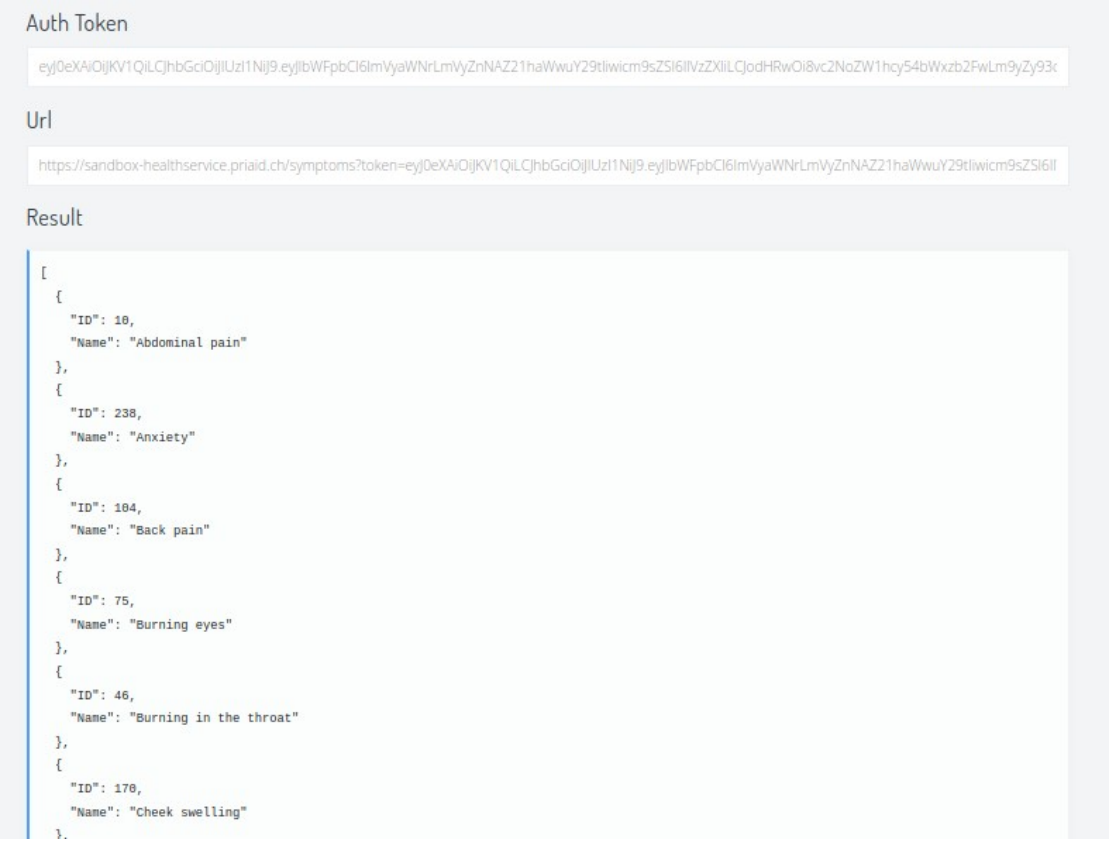


Figura 22 – Consulta de Sintomas- APIMedic. Fonte: apimedic.com (2021a)

A ideia principal para o consumo desta API é a utilização baseada no sintoma do paciente. Ele irá informar as dores ou sintomas de uma possível doença, a API retornará mais opções específicas para melhorar a consulta da doença e na nova requisição, retornará com o possível diagnóstico, acrescido de médico recomendado.

```
{
  "Issue": {
    "ID": 53,
    "Name": "Constipation",
    "Accuracy": 45,
    "Icd": "K59.0",
    "IcdName": "Constipation",
    "ProfName": "Obstipation",
    "Ranking": 2
  },
  "Specialisation": [
    {
      "ID": 14,
      "Name": "Gastroenterology",
      "SpecialistID": 0
    },
    {
      "ID": 15,
      "Name": "General practice",
      "SpecialistID": 0
    },
    {
      "ID": 19,
      "Name": "Internal medicine",
      "SpecialistID": 0
    }
  ]
},
```


Figura 23 – Consulta de Diagnóstico- APIMedic. Fonte: apimedic.com (2021a)

Com as consultas por sintomas, informando a idade e o sexo, o retorno traz a listagem com possíveis doenças em *Name*, e com os especialistas e médicos recomendados em *Specialisation*. O retorno do JSON é claro e simples de ser utilizado, os dados informados serão de grande valia e por conta disso a APIMedic também será utilizada no processo de criação do CUIDA.

4.2.5 SPRINT 05 – Comunicação com API e tela de HomePage.

4.2.5.1 - Planejamento

Como forma de manusear o conteúdo dos artigos que estão sendo retornados em HTML na API MyHealthfinder (tópico 5.2.4.2), em uma próxima *sprint* (Trabalhos Futuros, tópico 7) será decidido como tratar este tipo de retorno em um aplicativo. Por hora nesta mesma Sprint será feito:

- As classes que irão comunicar com a API;

- Bem como o modelo de retorno.
- Retorno dos dados até a HomePage.

Com esta última *sprint* do projeto, o foco será nas informações comunicadas com a MyHealthfinder, por conta disso, o restante do *backlog*, bem como as atividades pendentes deste projeto, estarão mencionadas no tópico 7.

4.2.5.2 - Desenvolvimento

Como informado no tópico 5.2.3.2 no Desenvolvimento de outra sprint, o processo de leitura de uma tela no Flutter começa com o seu Binding. Nele é feito a chamada de controllers e processos que irão compor a página com dados trazidos da API.

Na tela de HomePage, sua controller é carregada após o binding a chamar. É feita a declaração das variáveis globais da classe e em seguida é feita uma sobrescrita de método com o *@override*, o qual irá iniciar o método abaixo antes de todos os outros. Após isso é chamado um método do dart o qual se chama onInit(), que de forma assíncrona, irá carregar antes que todos os outros métodos da controller.

Figura 24 – Classe HomeController – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Ainda na própria controller, de forma assíncrona é feita a chamada do método getArticles(), que declara com 3 classes sendo duas abstratas, retornará futuramente uma lista de Resource.

Entrando no bloco após declarar na linha 22 uma variável local, é esperado do retorno de uma nova chamada de método na linha 24, homeRepository.getArticleList(). Aqui uma classe abstrata é declarada e depois chamada por uma classe de repositório.

Figura 25 – Classe HomeRepository – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Nesta classe é onde é feita a comunicação com a API. Percebe-se que na linha 9 é feita a declaração de uma variável global que importará os métodos da biblioteca Dio, que dentro dela é importada as configurações bases criadas para esta conexão.

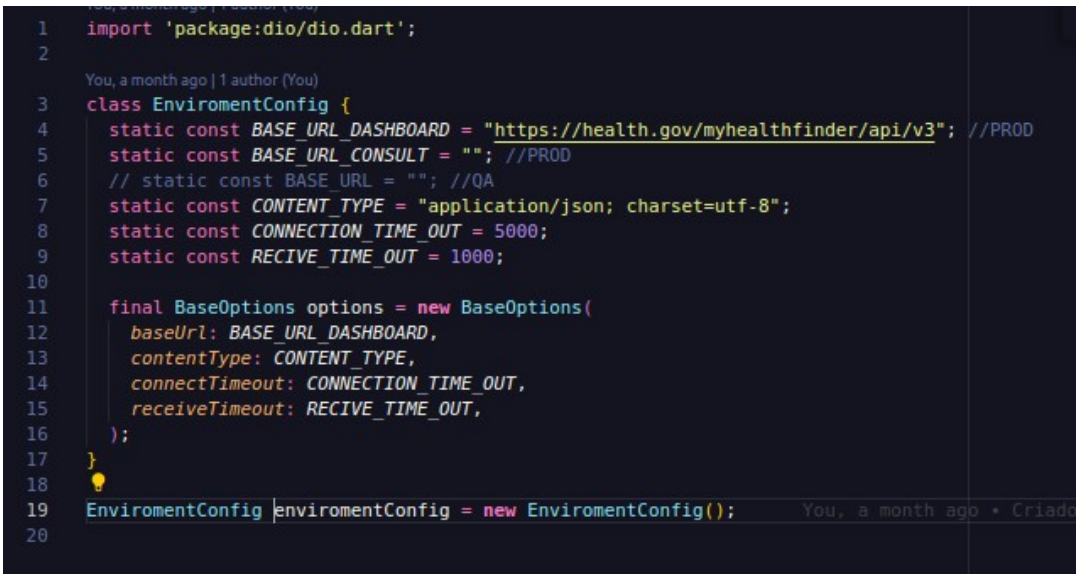


Figura 26 – Classe EnviromentConfig – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Na EnviromentConfig é possível ver o tempo de requisição preestabelecido, a *url* base que a API possui, tempo de resposta e conectividade.

Retomando até a figura de HomeRepository, é possível notar que na linha 14 é feita a tentativa de conexão com a variável declarada do *dio* com a *url* de caminho da consulta. Após o retorno os dados, o *response.data* é declarado dentro de ArticleModel que irá mapear os dados.

Figura 27 – Classe ArticleModel – CUIDA. Fonte: Elaborado pelo Autor (2021a)

Nesta sequência, os dados serão passados de classes de modelo o qual serão tratados e retornarão do HomeRepository até a HomeController e assim, após verem que os dados foram modificados na variável declarada, a classe Obx irá tratar de carregar a listagem que veio de getArticles() em HomeController.

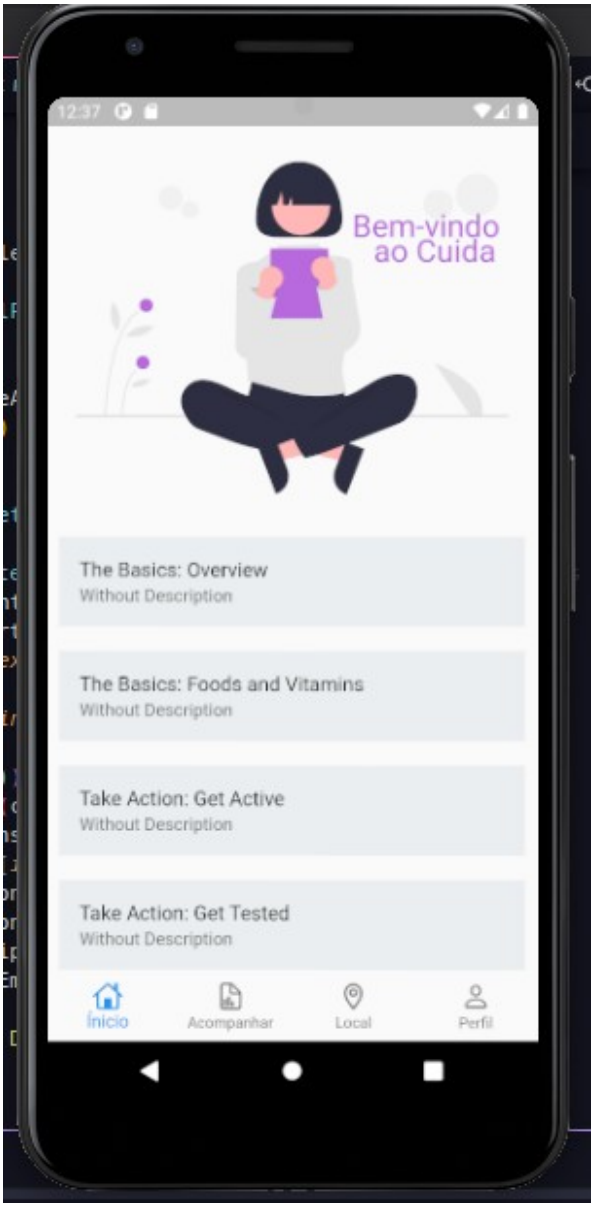


Figura 27 – Tela de HomePage – CUIDA. - Fonte: Elaborado pelo Autor (2021a)

É possível ver o resultado na image acima, os cards com fundo pré-selecionado de cores pesquisados na primeira sprint, bem como a imagem importada em assets. Esta listagem, traz o título dos artigos que nas sprints futuras irão, após ser tocado, abrir os detalhes do artigo em questão.

5 CONSIDERAÇÕES FINAIS

Com o desenvolvimento deste projeto, pode-se ter noção sobre vários pontos.

O primeiro deles é da real necessidade de uma equipe bem estruturada para gerenciar um projeto. Por ser uma única pessoa desenvolvendo o sistema, pude notar seguintes aspectos referente ao projeto.

- A necessidade de um arquiteto de sistemas, bem como um gerente de projetos para compor o CUIDA.

- Estruturar o *frontend* sem noção do que vai programar é perder tempo refatorando código.

- Um designer é mais necessário do que se imagina.

Este último ponto achei mais relevante. Diante da necessidade de se fazer o levantamento de requisitos em um projeto, mesmo já tendo noção das funcionalidades, e a estrutura e os requisitos que não deixam de ser necessários, um designer que sabe como funcionará o fluxo do sistema, trabalhando junto da equipe de projeto terá resultados de melhor qualidade, o layout será melhor projetado e assim resultará numa melhor agradabilidade do usuário, bem como a equipe de *frontend* ou mobile, terá como se guiar quando algo está bem desenhado e levantado..

Por ser um desenvolvedor frontend e mobile, percebi também a real necessidade de documentação de uma API. Trabalhar às cegas e efetuar consultas em APIs não documentadas corretamente, ou iniciar a projeção de um sistema em endpoints que estão repetidos, só mostra a falta de pessoal qualificado no setor, bem como a falta de documentação correta e estruturada de um sistema que pode escalar ou não diante da necessidade dos stackholders. Mostra-se cada vez mais necessário gerentes de projetos que permitam a si mesmos, entenderem que a parte primordial de um projeto é a de levantamento de requisitos, para que a equipe o qual foi designada para determinado projeto, possa efetuar os *sprints* com mais agilidade e coerência.

Trabalhar em um projeto como este levanta questões até mesmo profissionais que, durante o processo de criação dele, me mostrou a necessidade de aprender como funciona o backend, a criação de uma API e a gerência do banco de dados.

Estarei ciente que diante da continuidade do CUIDA, projetarei a possibilidade de aprender backend para assim utilizar algo próprio, a fim de ter uma melhor resposta e ciência dos dados consultados.

6 TRABALHOS FUTUROS

O *backlog* deste projeto não foi finalizado; tão pouco foi levado até a metade do previsto para ele. Com o intuito de levá-lo a um patamar de aplicativo publicável em uma loja, tanto Apple quanto Google, percebi que a principal tarefa a se seguir seria finalizar a comunicação, bem como o layout das telas principais.

Portanto, para trabalhos futuros, mediante a um backlog amplo e pouco detalhista fica pendente:

- Internacionalização. O aplicativo atualmente faz a comunicação com APIs americanas, estas, possuem linguagem para outros idiomas fora o português, portanto, a necessidade de trazer o aplicativo para o público brasileiro, traduzindo a resposta de forma automática e ainda assim padronizada é uma tarefa para o trabalho futuro.
- Usufruir da comunicação da APIMedic para consultas de sintomas e doenças. Atualmente, no sprint que finalizou este projeto, a comunicação com a APIMedic não foi iniciada. Para o CUIDA é a parte principal do projeto. Esta tela será desenhada e pensada na melhor tratativa para o usuário.
- Geolocalização. Como mostrado no Menu principal do projeto a aba de localização irá consultar a geolocalização do usuário, a ideia é mostrar as clínicas próximas cadastradas no Google Maps
- Por fim, mas não menos importante a gerência da conta do usuário. Este poderá informar os sintomas, quadro de melhoras e cadastro de informações para que futuramente em uma versão 2, possa ser utilizado da melhor forma possível os dados armazenados (com a permissão do usuário), a fim de levar a comodidade e a segurança sobre sua saúde.

REFERÊNCIAS

APIMEDIC. **ApiMedic.** Disponóvel < https://apimedic.com/news > Acesso em 28 de nov 2021.

BALTIERI, André. **Flutter: Por onde começar?** Disponível: < https://balta.io/blog/Flutter-por-onde-comecar > Acesso em 24 de out. 2021

CANALTI. **XML e JSON** Disponível <https://www.canalti.com.br/programacao/xml-e-json-o-que-e-semelhancas-diferencas-utilizacao/>Acesso em 10 de nov de 2021.

FIA. **Scrum**: o que é e como aplicar a metodologia ágil para gestão? Disponível. < https://fia.com.br/blog/scrum/ > Acesso em 12 de nov. De 2021.

GUEDES, Marylene. **O que é Dart?** Disponível: <https://www.treinaweb.com.br/blog/o-que-e-dart#:~:text=O%20Dart%20%C3%A9%20uma%20linguagem,de%20scripts%20em%20p%C3%A1ginas%20web.&text=O%20Dart%20possui%20algumas%20variantes,a%20seu%20ambiente%20de%20execu%C3%A7%C3%A3o.> Acesso em 24 de out. 2021

LIMA, Thiago. (Parte 1) **A anatomia de uma API RESTful** Disponível <https://thiagolima.blog.br/a-anatomia-de-uma-api-restful-80df2aca158e> Acesso em 10 de nov. 2021.

LEAL, Adson; LEAL, Angelo. **Scholar**: Desenvolvimento de um aplicativo móvel genérico de apoio acadêmico a estudantes em universidades. Florianópolis. 2019.

MARC. **StatelessWidget vs. StatefulWidget.** Disponóvel < https://www.Flutterclutter.dev/Flutter/basics/statelesswidget-vs-statefulwidget/2020/1195/> Acesso em 17 de nov 2021.

MYHEALTHFINDER. **Health.gov.** Disponóvel < https://health.gov/our-work/national-health-initiatives/health-literacy/consumer-health-content/free-web-content/apis-developers/api-documentation > Acesso em 28 de nov 2021.

REDHAT. **O que é API?** Disponível < https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces > Acesso em 10 de nov. 2021.