HEMOSHARE

APP PARA DOAÇÃO DE SANGUE
EM FLUTTER

ALUNOS - CTII 417

ANA JULIA OLIVEIRA DA SILVA

ANTONY NASCIMENTO DA SILVA

JULIA CARANO ANDRADE DA ROCHA

LUIGI FILIPE BONVENUTO

MARIA LUÍSA SOARES LEITE

YASMIN DE JESUS VERDE XAVIER

ORIENTADOR

MAURICIO NEVES ASENJO

SUMÁRIO

INTRODUÇÃO

O nosso projeto é um aplicativo *mobile* desenvolvido através da framework Flutter, utilizando a linguagem de programação Dart e os *packages*. Como IDE (ambiente de desenvolvimento integrado) foi utilizado principalmente o VS Code (Visual Studio Code) e o Android Studio para emular.

O aplicativo se chama **HemoShare** e tem como tema a doação sanguínea, auxiliando no contato entre o doador e posto de doação, fornecendo informações necessárias para o procedimento,

OBJETIVOS

1. Objetivo geral

Promover o interesse e a informação em relação aos processos de transfusão e doação de sangue.

2. Objetivos específicos

- Identificar as carências e desafios na área de doação de sangue;
- Informar sobre acessos e critérios sociais e biológicos para o enquadramento como doador;
- Estabelecer a comunicação entre o hemocentro e o doador;
- Estimular o interesse da população mediante a relevância do procedimento;
- Desenvolver uma aplicação *mobile* que coopere com as etapas de doação de sangue.

APLICAÇÕES UTILIZADAS

1. FLUTTER

a. Sobre o Flutter

O Flutter é um framework multiplataforma desenvolvido pela Google e baseado em Dart,que combina as vantagens de linguagens robustas como Java e adaptabilidade do JavaScript. Os blocos de código representam widgets que formarão a interface do usuário.

O Dart é uma linguagem client-side e estruturada segundo o paradigma orientado a objetos, como o Java, buscando ser flexível para servir como base para diversos frameworks de desenvolvimento de aplicativos, embora até o momento o Flutter seja o único que utilize.

Sua primeira versão possuía outro nome, “Sky”, e surgiu com a ideia de construir melhores interfaces para o *mobile*. Lançado inicialmente em 2014 e apresentado ao público em 2015, estreou no Google IO 2017, já integrado ao Firebase, o sistema de banco de dados unificado da Google. No Google IO 2018 foi lançada a versão 1.0
Ele não traduz o código para o respectivo elemento, mas implementa através de um motor de renderização próprio. Possui compilados em ARM nativ

Possui Hot Reload, ou seja, testa em tempo real, não sendo necessário aguardar a compilação para visualizar o resultado
.
A partir de 2019 se tornou uma framework portável para *mobile*, web, desktop e embarcados, sendo multiplataforma. Em 2022 o Google lançou o Flutter 3, a versão mais recente da framework que abre espaço para a criação de programas para Linux, macOS e suporte para o Apple MI.

https://doutbox.com.br/blog/flutter-3-o-que-ha-de-novo#:~:text=Em%20maio%20deste%20ano%20(2022,framework%20para%20desenvolvimento%20de%20aplicativos.
https://www.alura.com.br/artigos/flutter

b. Pré-requisitos

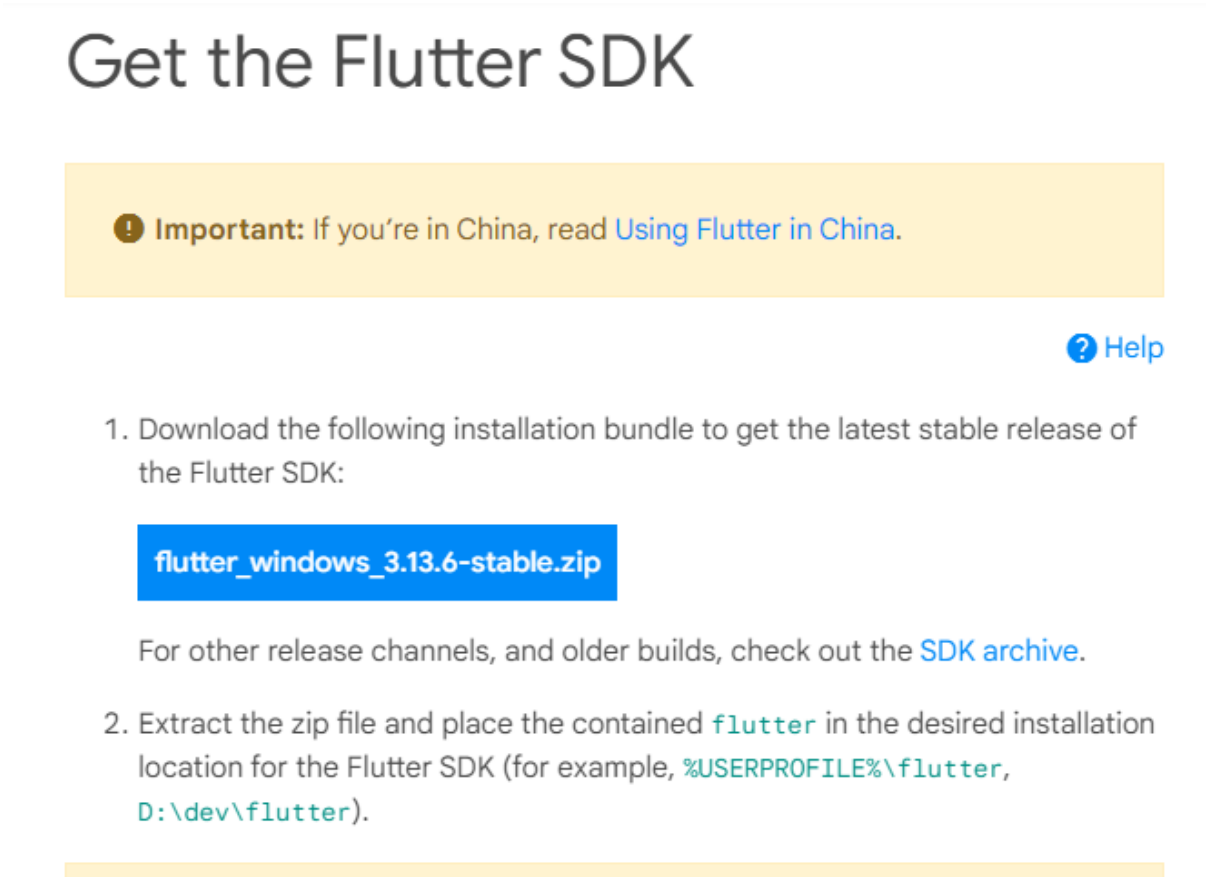
- Windows 7,8 ou 10
- Git instalado no sistema

c. Instalação

1. Baixar a SDK do Flutter

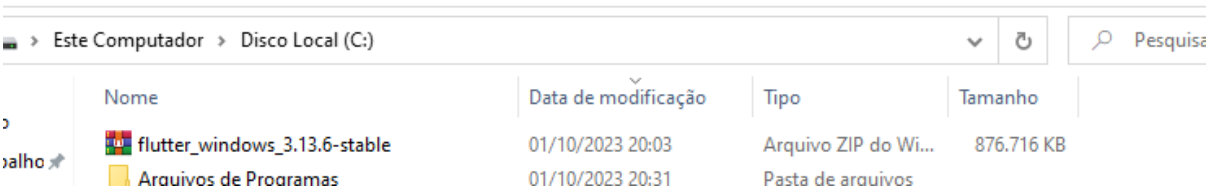
Acesse o site oficial e clique em “Download” para baixar a SDK[1] do Flutter como um arquivo .zip. É importante ressaltar que a SDK do Dart vem junto na instalação, logo não é necessário instalá-lo separadamente

[1] SDK é kit de desenvolvimento de software, um conjunto de ferramentas oferecidas para os desenvolvedores como depuradores, compiladores e bibliotecas para a criação de códigos



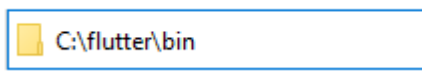
2. Extraia no disco local

Uma vez que o arquivo está baixado, transfira para o Disco Local e extraia



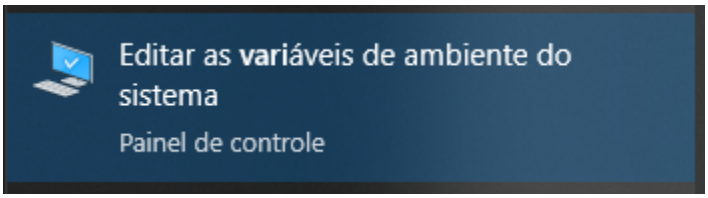
3. Copie o caminho da pasta "flutter"

Para que os comandos sejam realizados diretamente pelo terminal é necessário inserir o diretório do "flutter/bin" nas variáveis de ambiente do Windows, PATH [2]. Em seguida, abra o editor de variáveis de ambiente do sistema

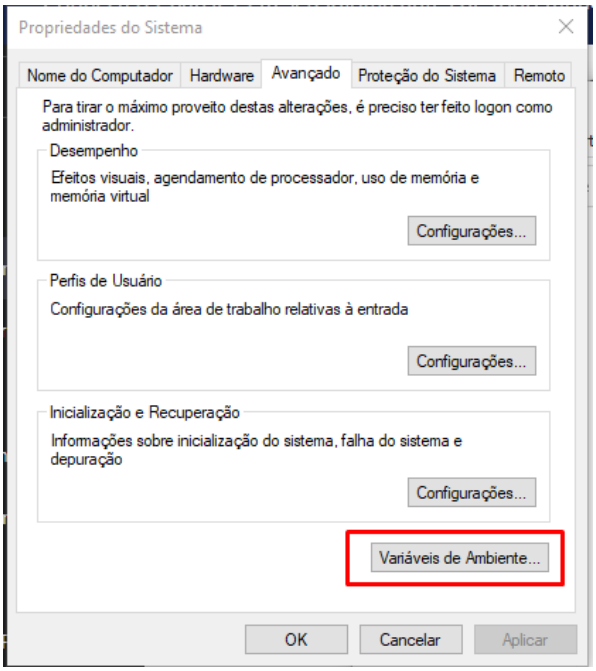


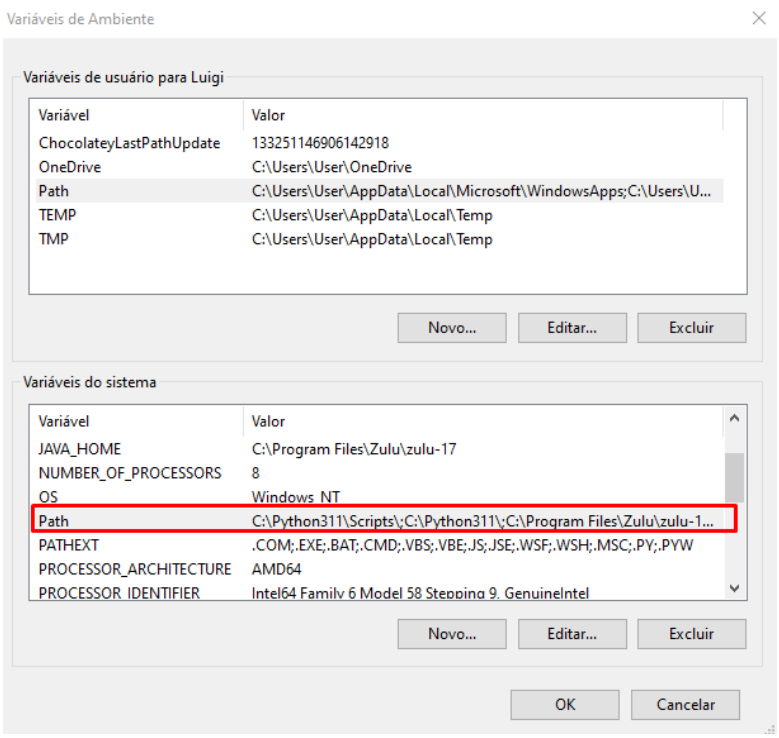
[2] PATH é uma variável de ambiente no Windows que especifica uma lista de diretórios onde os arquivos executáveis (.exe) estão localizados. Ao inserir um comando no terminal (cmd) o sistema busca no diretório um executável com o nome correspondente, caso não seja encontrado, busca no PATH.

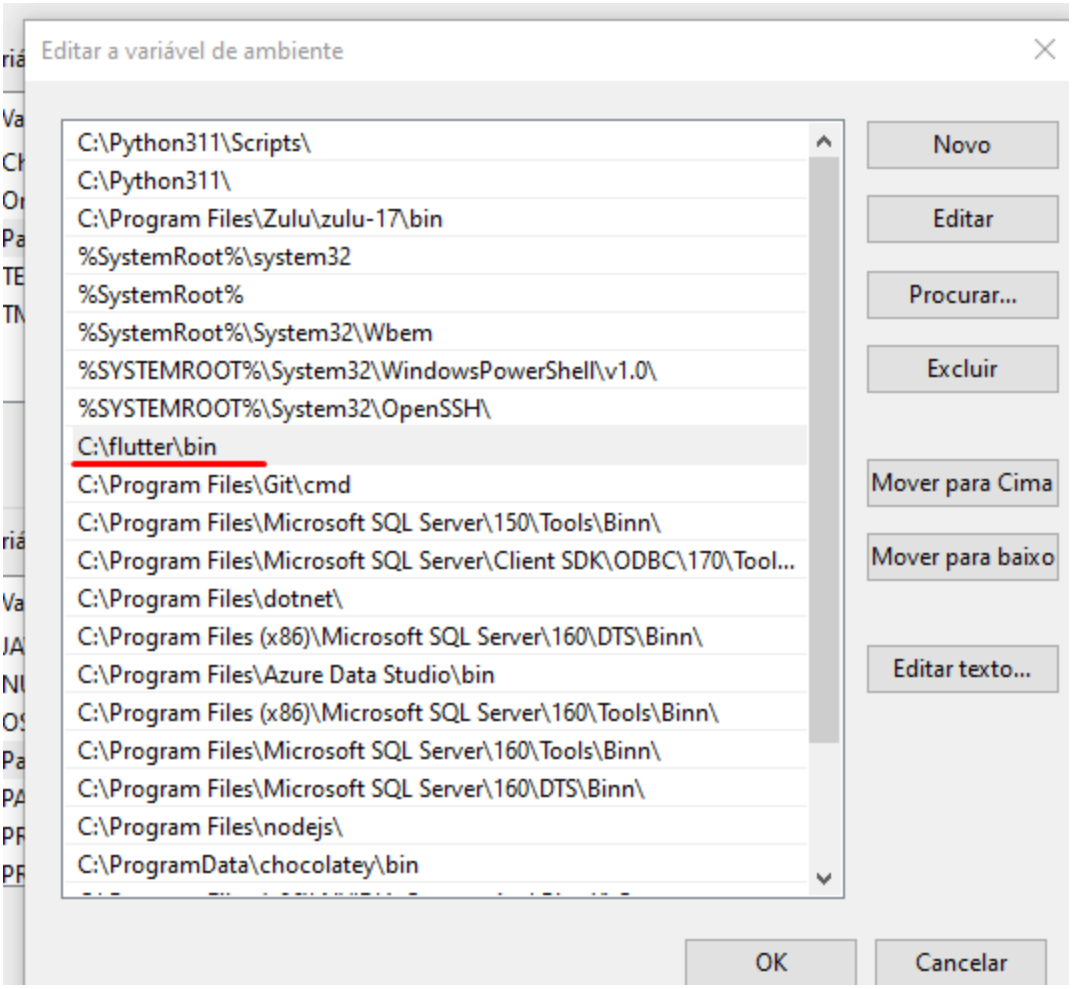
Adicionar o Git ao Path permite que os comandos sejam realizados a partir do próprio terminal, sem precisar estar no diretório onde ele foi instalado



4. Adicione o caminho da pasta flutter na variável PATH

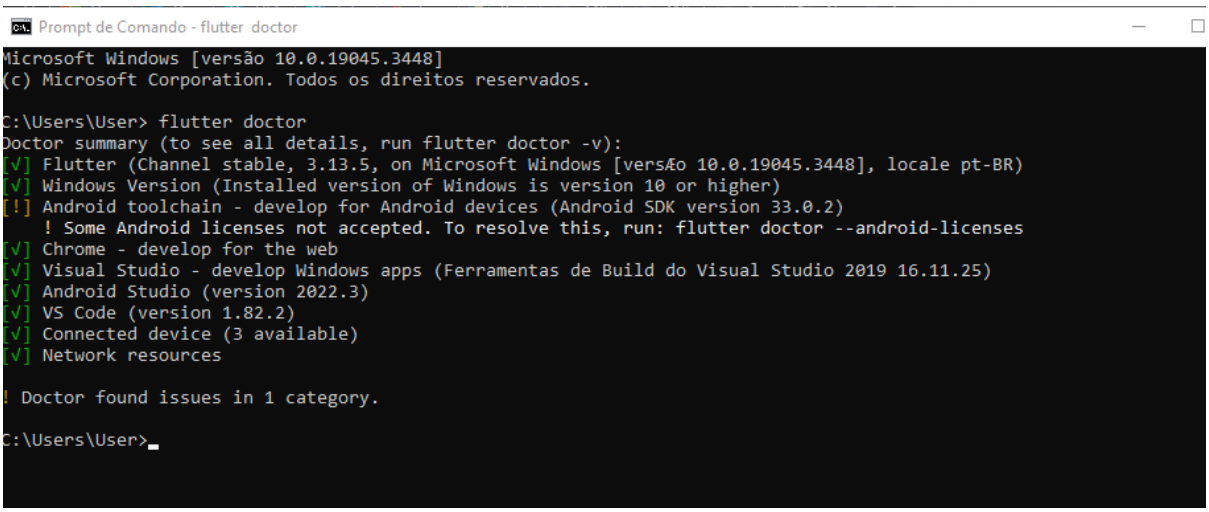






5. Verifique se a instalação foi feita corretamente

Realizando o comando -flutter doctor é possível verificar se a instalação foi feita corretamente



Pronto! O Flutter está instalado. Em seguida, será explicado a instalação do editor para utilizá-lo, no caso o VS Code

2. VISUAL STUDIO CODE

a. Sobre o Visual Studio Code

O Visual Studio Code é um editor de código aberto desenvolvido pela Microsoft disponível para Windows, Mac e Linux. Não deve ser confundido com o Visual Studio, também criado pela Microsoft e dedicado ao .NET Framework e linguagens como C e C#.

O VS Code possui funcionalidades mais simples e é mais customizável, podendo editar programas e script de diferentes linguagens de acordo com as extensões baixadas.

b. Pré-requisitos

- Sistema Operacional: Windows 10, 8.1 ou 7 (32 ou 64 bit)
- Espaço em disco: 50 GB de espaço livre recomendado
- RAM: 8 GB de RAM recomendado (mínimo de 4GB)

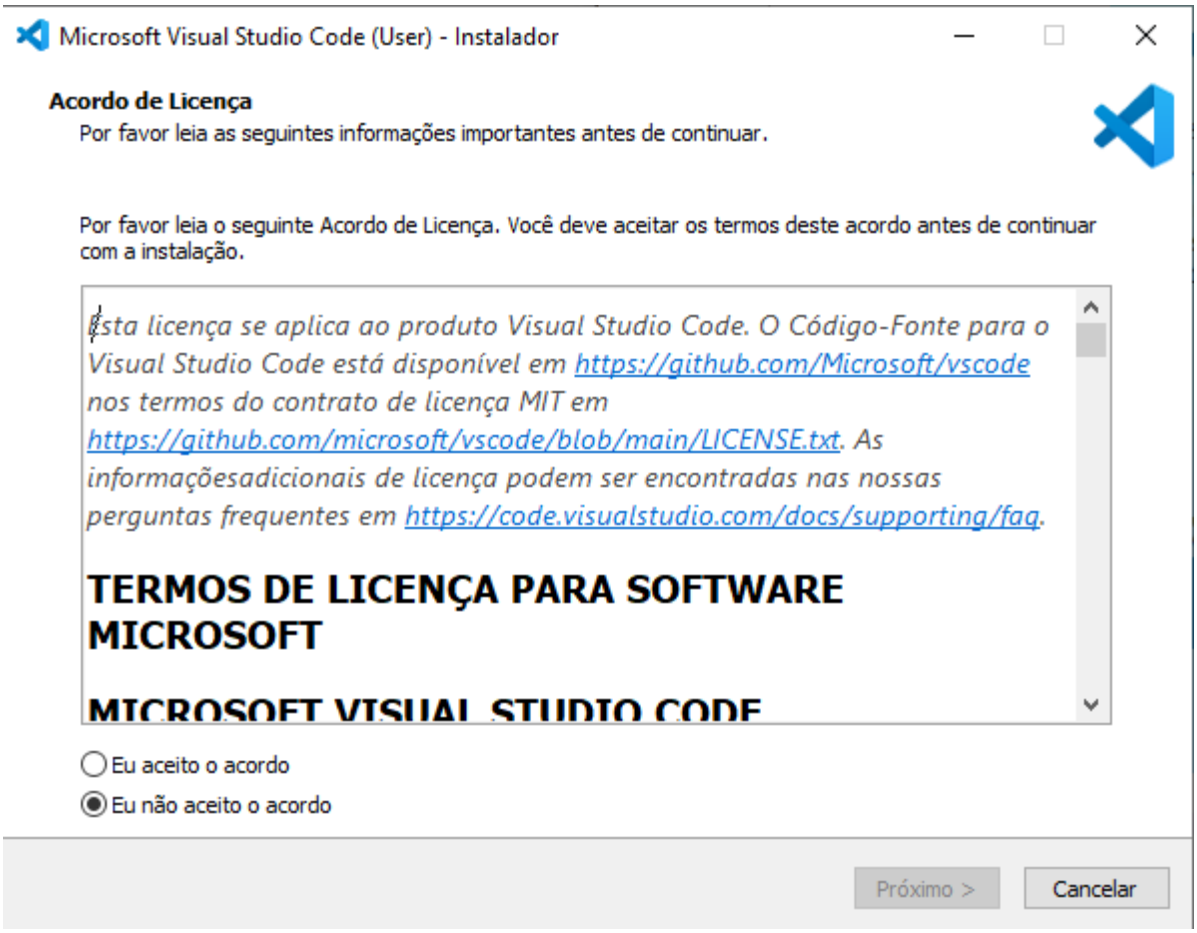
c. Instalação

1. Download do Instalador

No site https://visualstudio.microsoft.com/pt-br/ baixar o instalador do Visual Studio Code

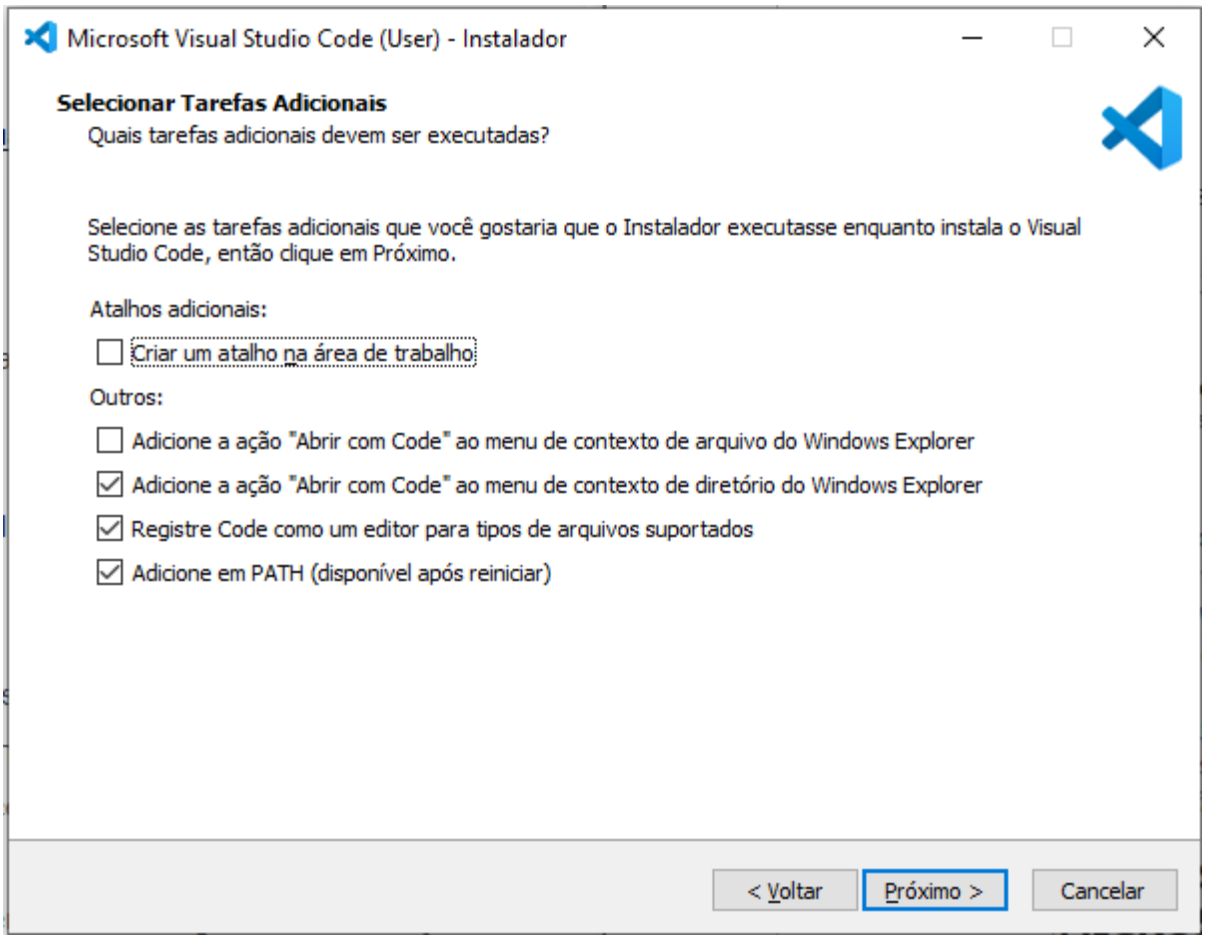
2. Executar o Instalador

Abrir o arquivo baixado do instalador e definir a pasta em que será instalado, por definição uma subpasta na AppData



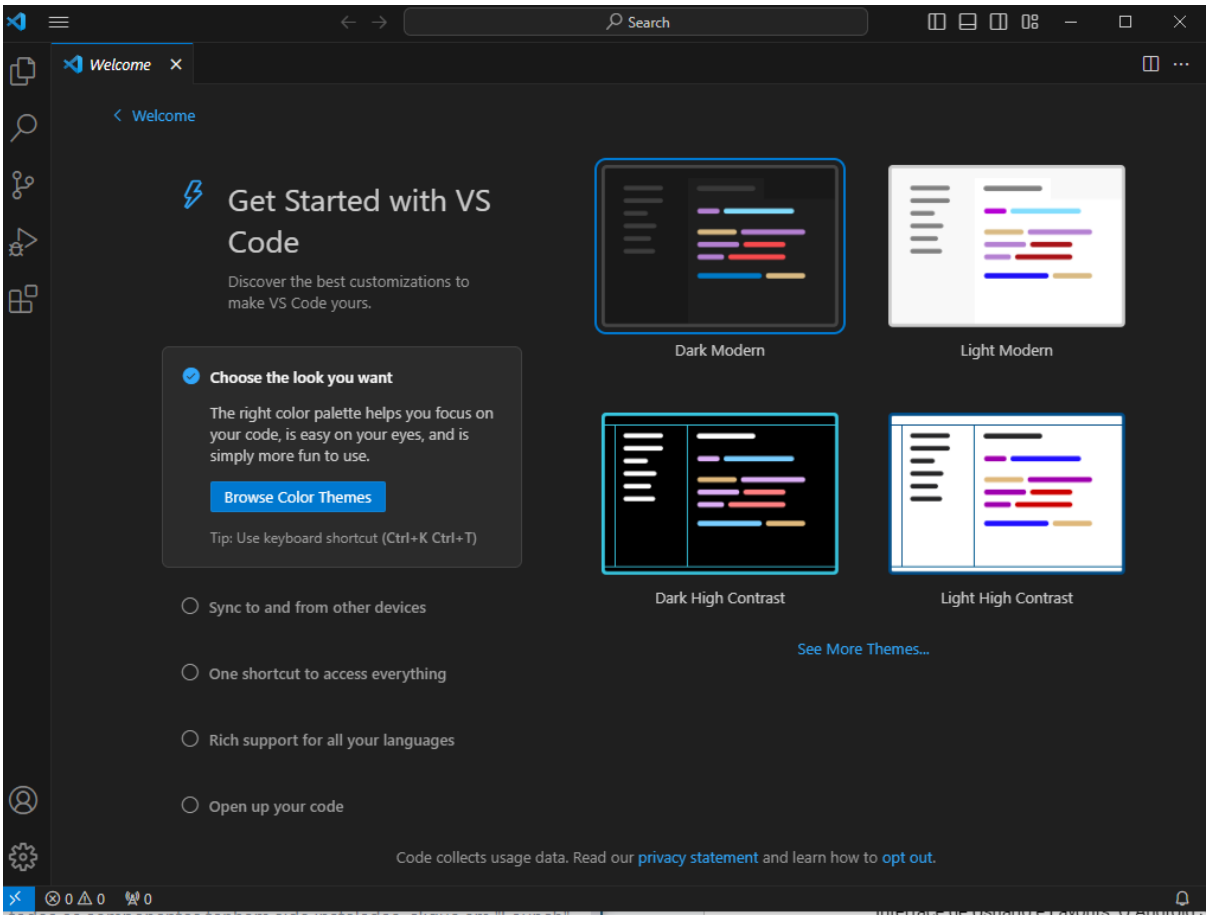
3. Configurar a instalação

Definir as opções de “Tarefas Adicionais” segundo a necessidade de instalação, nesse caso, as já selecionadas são suficientes



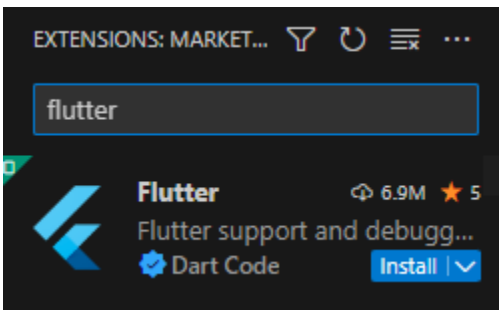
4. Configuração inicial

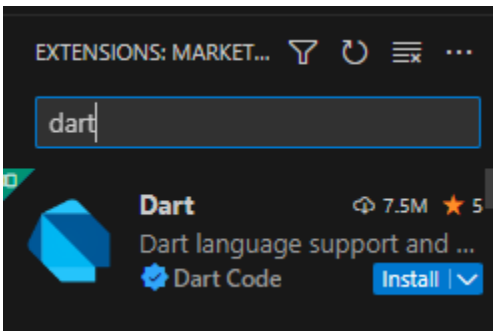
Definir o tema da IDE (branco ou escuro) e sincronizar com outros dispositivos são as opções disponíveis para essa etapa



5. Baixar as extensões de Flutter e Dart

Na aba de “Extensões” (ou apertando Ctrl+Shift+X), pesquisar “Flutter” e “Dart” e instalar ambas as extensões





Pronto! O seu Visual Studio Code já está instalado e devidamente configurado

3. GIT
4. FIREBASE
a. Sobre o Firebase

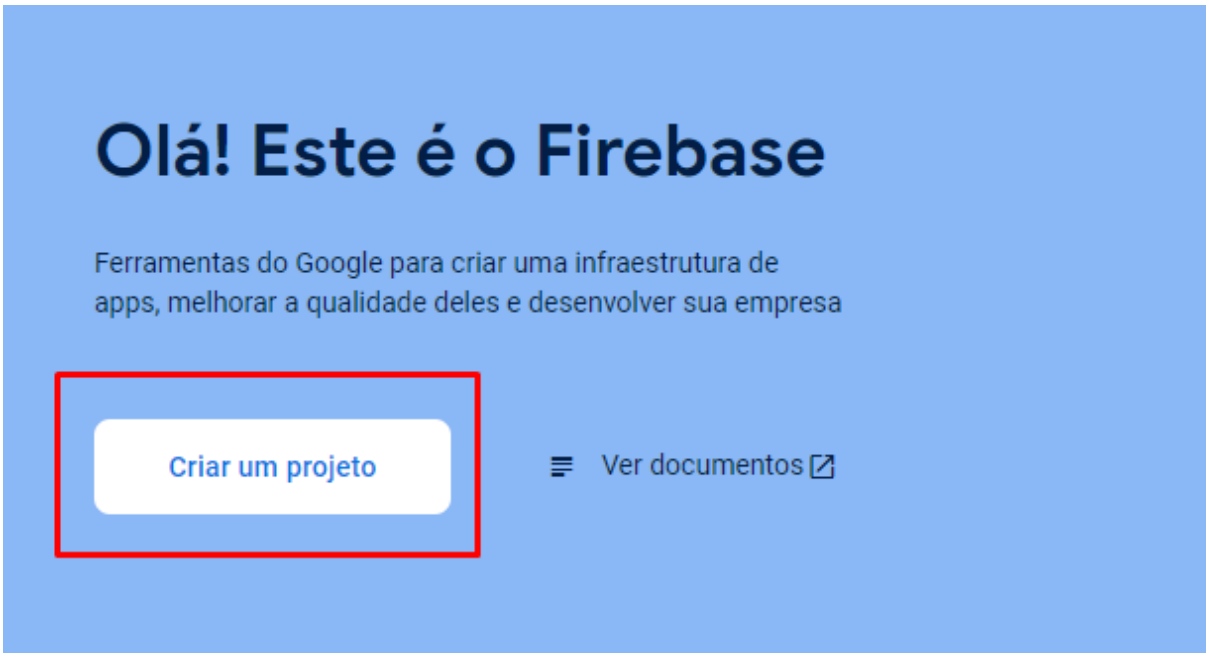
O Firebase é uma plataforma de desenvolvimento de aplicativos móveis e da web oferecida pelo Google que fornece uma variedade de ferramentas e serviços para ajudar os desenvolvedores a construir, melhorar e expandir seus aplicativos de forma eficiente. Dentre seus recursos estão o Firebase Authentication, o Firebase Cloud Storage, o Firebase Cloud Functions, Firebase Hosting, Firebase Analytics, etc.

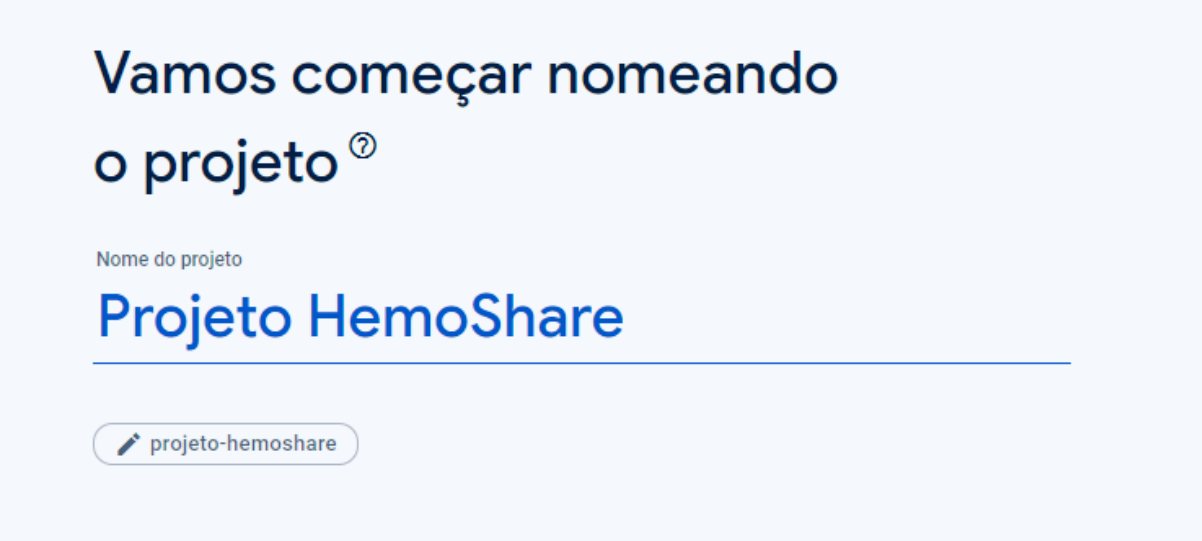
b. Configuração

1. Criar uma conta no Firebase

Essa conta pode ser a mesma conta google do usuário, devendo apenas acessar o site do Firebase: https://firebase.google.com/?hl=pt.

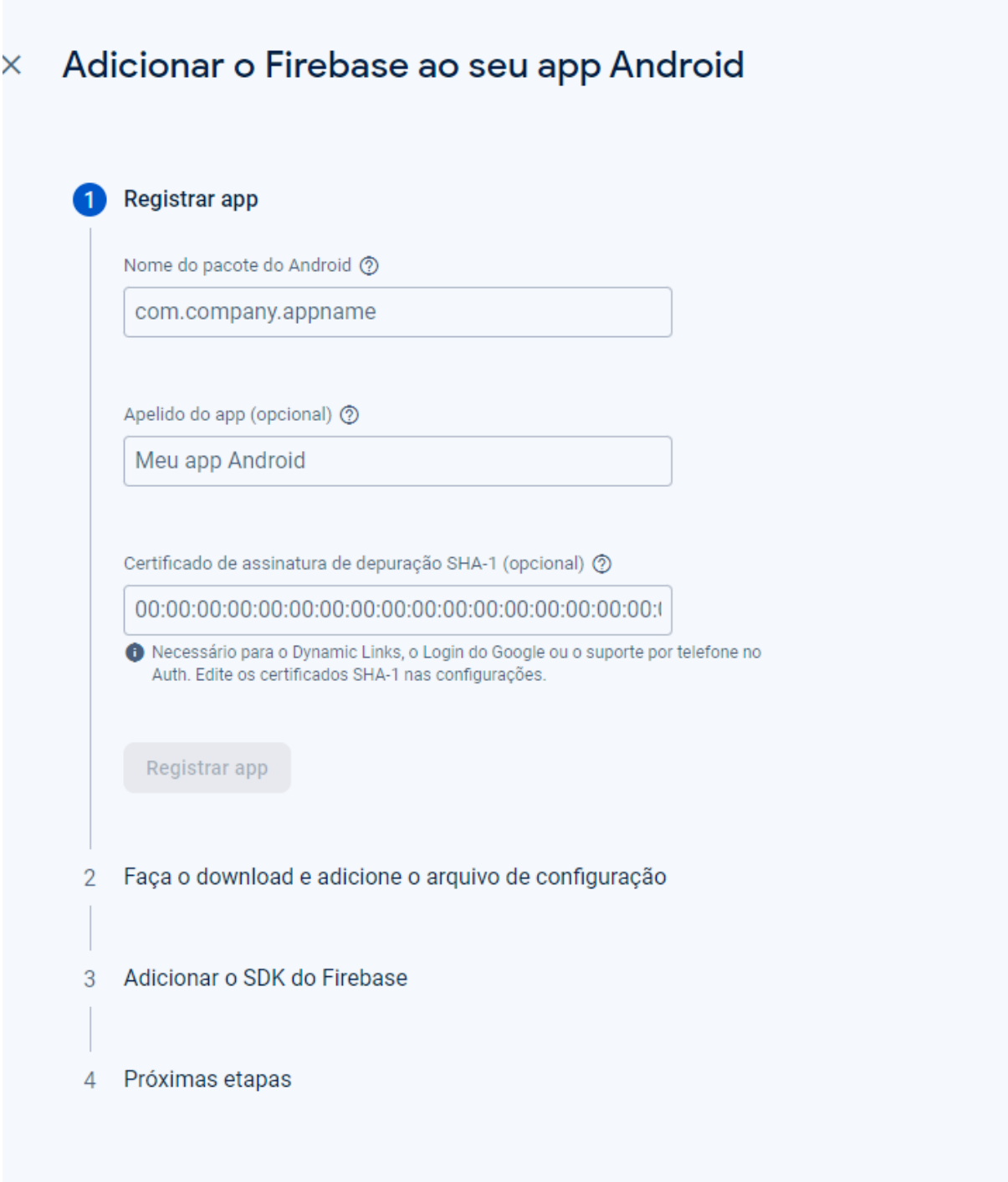
2. Criar um projeto





Na página do Firebase, deve criar um projeto e nomeá-lo de acordo com o nome estabelecido.

3. Adicionar um aplicativo



Seguindo o passo a passo do site, é possível adicionar e integrar com o aplicativo Flutter, para utilizar suas funcionalidades

4. Integrar com o código do aplicativo

```
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:hemoshare/arquivos/classes.dart';
import 'package:hemoshare/screens/redpoints.dart';
import 'package:provider/provider.dart';
import 'carregamento.dart';
import 'package:intl/date_symbol_data_local.dart';
```

Após realizar toda a configuração, é necessário importar os **packages** do Firebase para que seja possível acessar suas classes e funcionalidades.

a. Sobre o Git

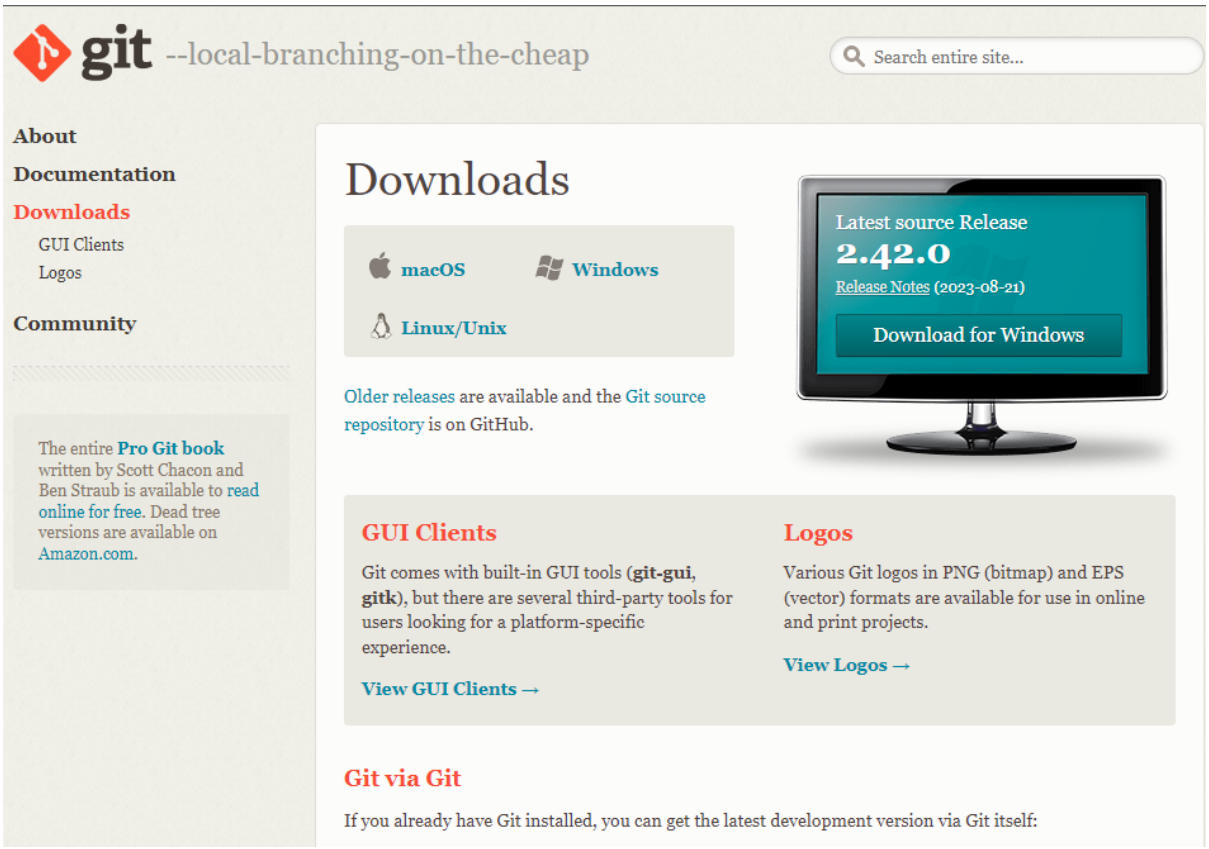
O Git é um sistema de controle de versão gratuito e open-source distribuído criado por Linus Torvalds em 2005, utilizado para rastrear alterações em arquivos e coordenar o trabalho entre múltiplos desenvolvedores de um projeto. É fundamental para fluxos de trabalho de desenvolvimento de software e permite ter versões distintas do mesmo código.

Ele é utilizado conjuntamente ao Flutter justamente para que possua um histórico completo das mudanças feitas ao longo do desenvolvimento do projeto, permitindo a restauração de versões anteriores em caso de erros que prejudiquem o código

b. Instalação

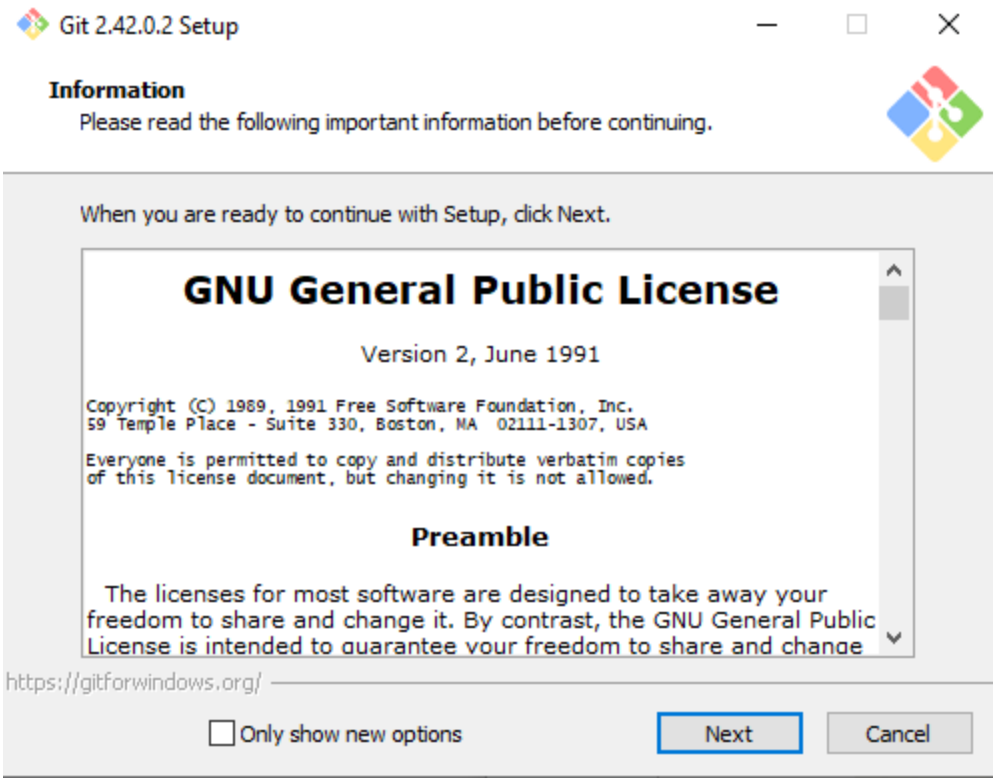
1. Baixar o Instalador

Acesse o site oficial (https://git-scm.com/downloads) e baixe a versão para o seu sistema operacional, geralmente Windows



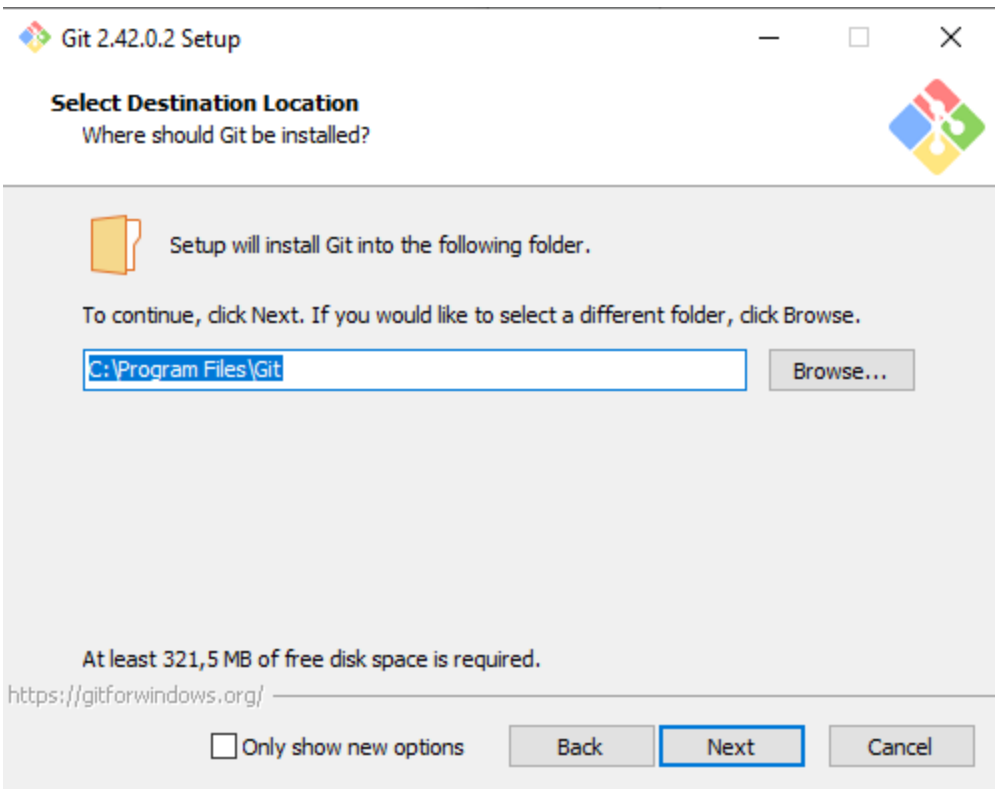
2. Executar o Instalador

Em seguida localize o arquivo baixado e inicialize o instalador, ele abrirá a documentação como página inicial



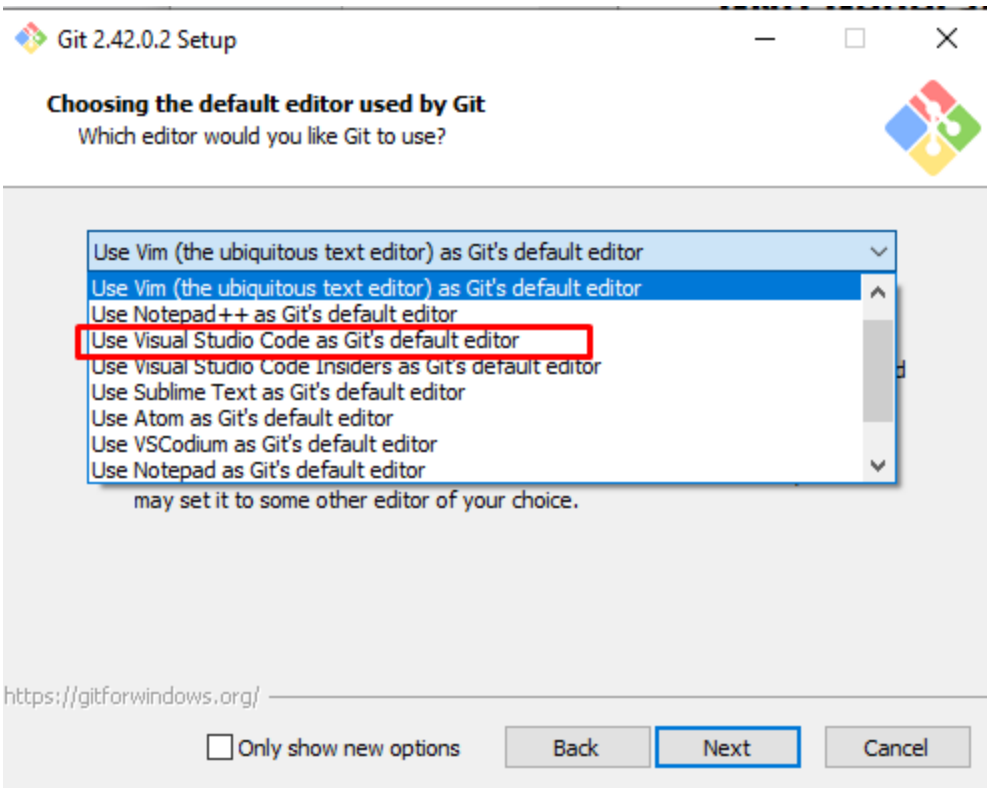
3. Definir o local de instalação

Determine em qual pasta o Git será instalado, de preferência no disco local



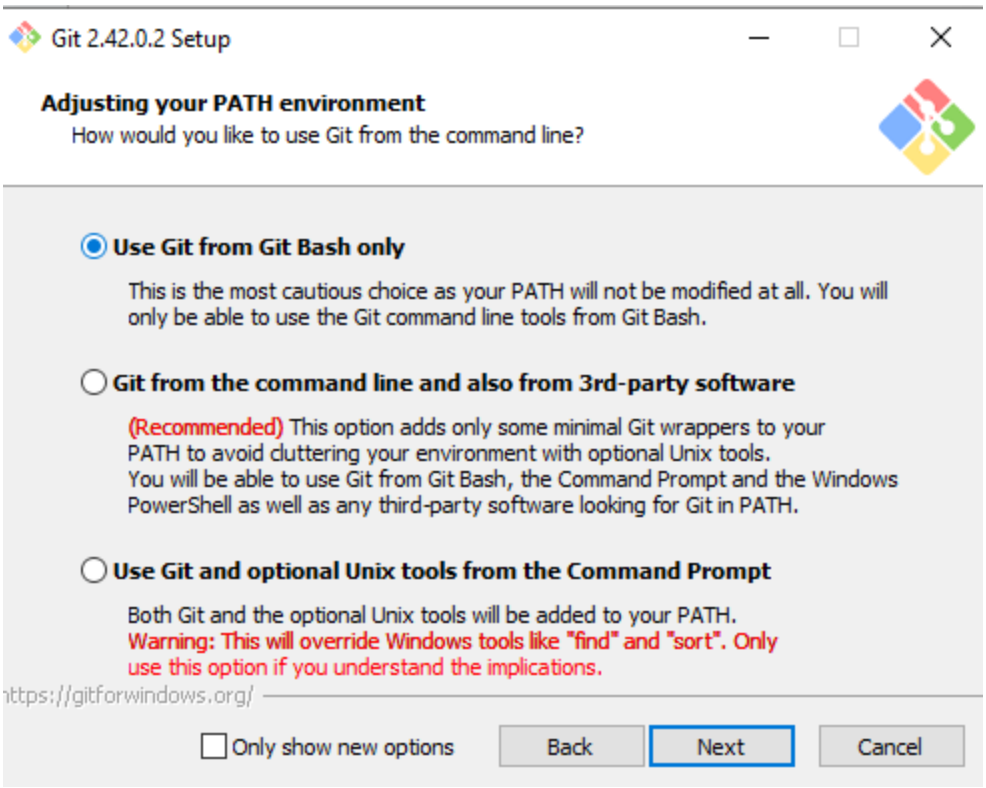
4. Determine o editor padrão

O editor de texto padrão do Git por natureza é o Vim, porém se você já possui o VS Code instalado (o processo também está neste documento), selecione a opção que o define como editor *default*



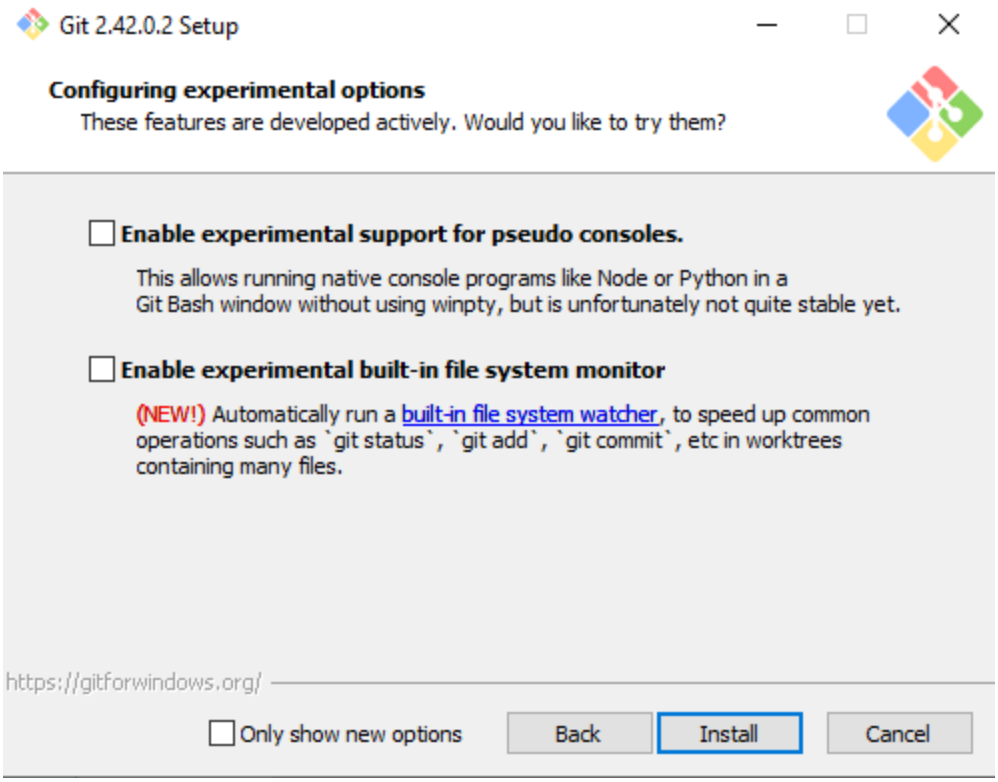
5. Configurar o PATH

Caso queira inserir o Git ao Path, marque a segunda opção, contudo a primeira é mais adequada para os iniciantes e que não pretendem realizar os comandos do Git diretamente do terminal



6. Prossiga até a instalação

Siga as definições padrão até chegar na última página, onde poderá instalar por definitivo o Git. Clique no botão instalar



Pronto! O Git está instalado em seu sistema e pode ser acessado pelo Git Bash (terminal próprio) ou pelo cmd (caso o Path tenha sido definido)

5. ANDROID STUDIO

a. Sobre o Android Studio

O Android Studio é uma IDE (ambiente de desenvolvimento integrado) oficial para o desenvolvimento de aplicações Android. Desenvolvido pela Google, é um emulador rápido com inúmeros recursos e ferramentas que simplificam o processo de criação, teste e publicação de aplicativos móveis.

Possui um editor de código altamente funcional e personalizável, capaz de suportar linguagens como Java, Kotlin (linguagem voltada a apps Android) e XML (Extensible Markup Language, utilizada para criar documentos). Seu sistema de compilação é baseado em Gradle, uma ferramenta de automatiza o build e oferece uma interface gráfica para a criação de interfaces de usuário (UI) dos *apps*, além de ter um editor de código personalizável e ferramentas de compilação e depuração.
Por ser integrado ao Git, possui integração com o sistema de versões, tendo acesso a versões anteriores do mesmo programa

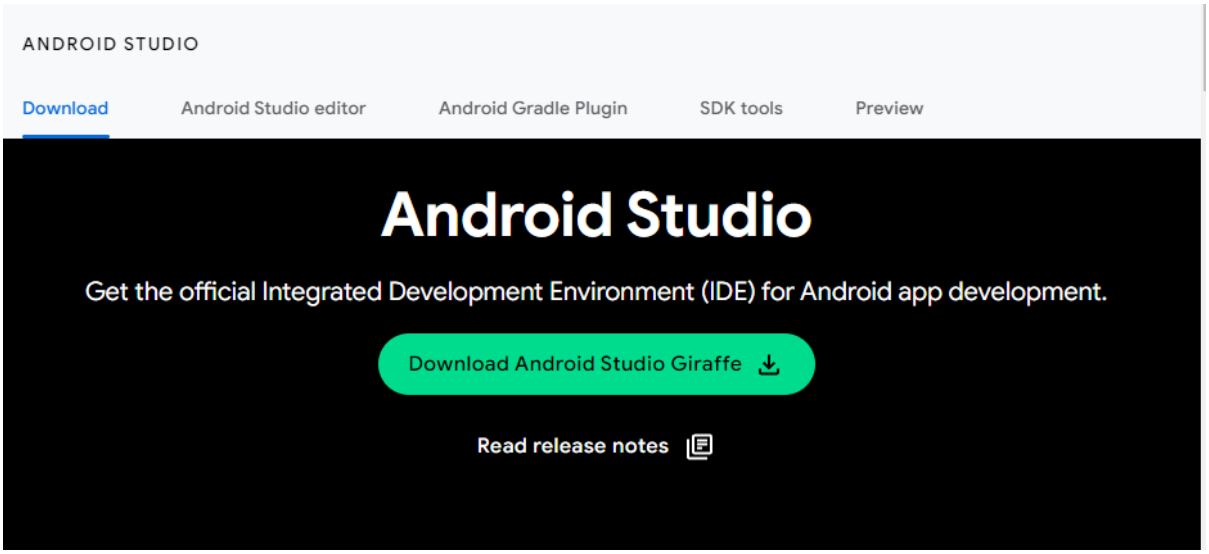
b. Pré-requisitos

- Sistema Operacional: Windows 10, 8.1 ou 7 (32 ou 64 bit)
- Espaço em disco: 4 GB de espaço livre recomendado
- RAM: 8 GB de RAM recomendado (mínimo de 4GB)

c. Instalação

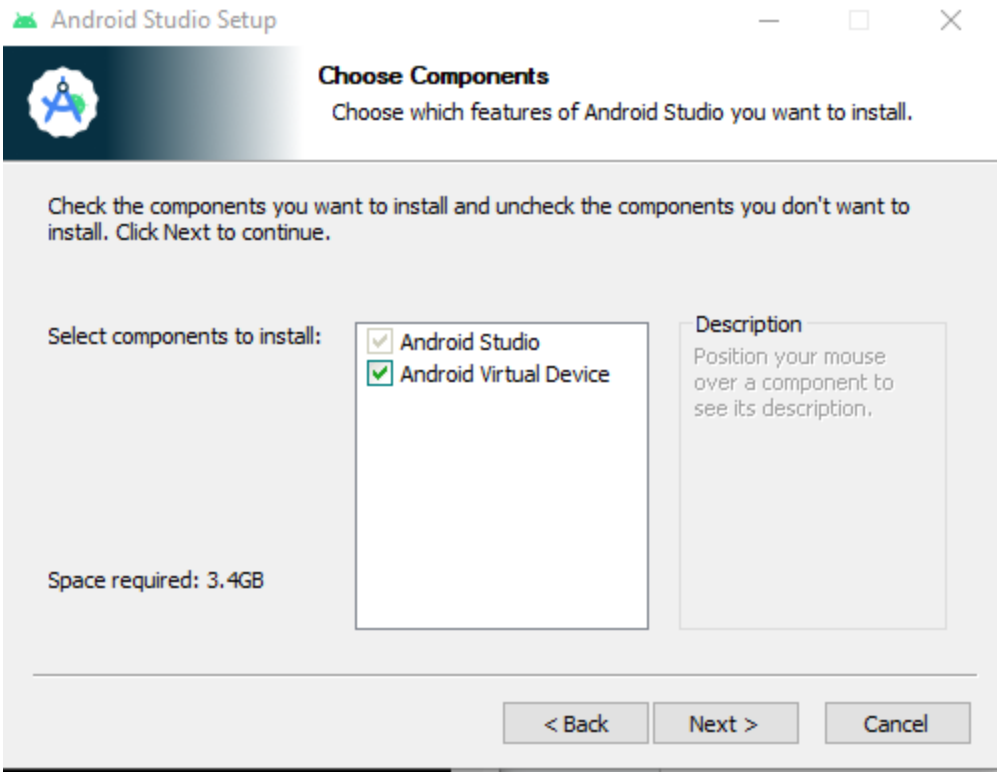
1. Baixar o Instalador

No site oficial (https://developer.android.com/studio) baixe o instalador do Android Studio. Após aceitar os Termos de Condição, o *download* começará



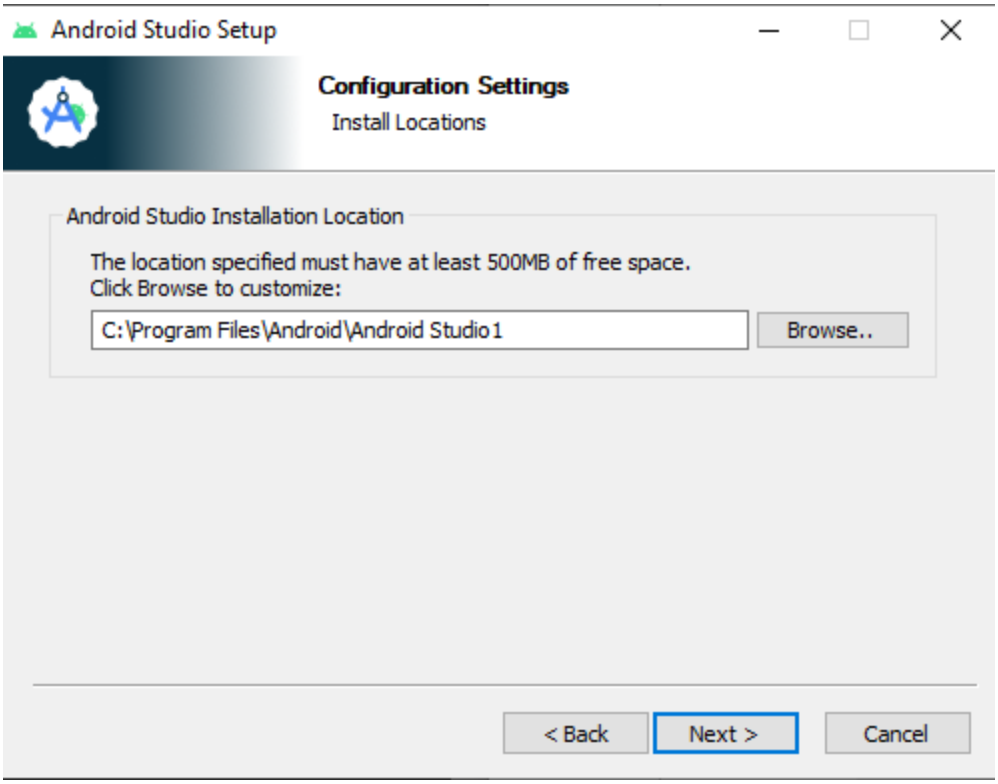
2. Abra o Instalador

Abra o arquivo .exe e escolha os componentes que serão instalados, no caso os já selecionados



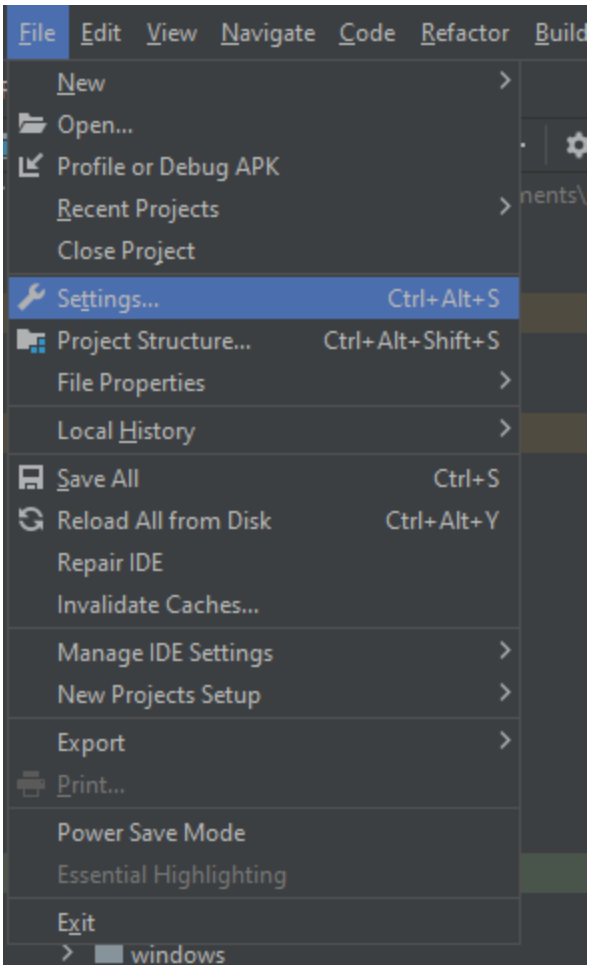
3. Escolha a pasta

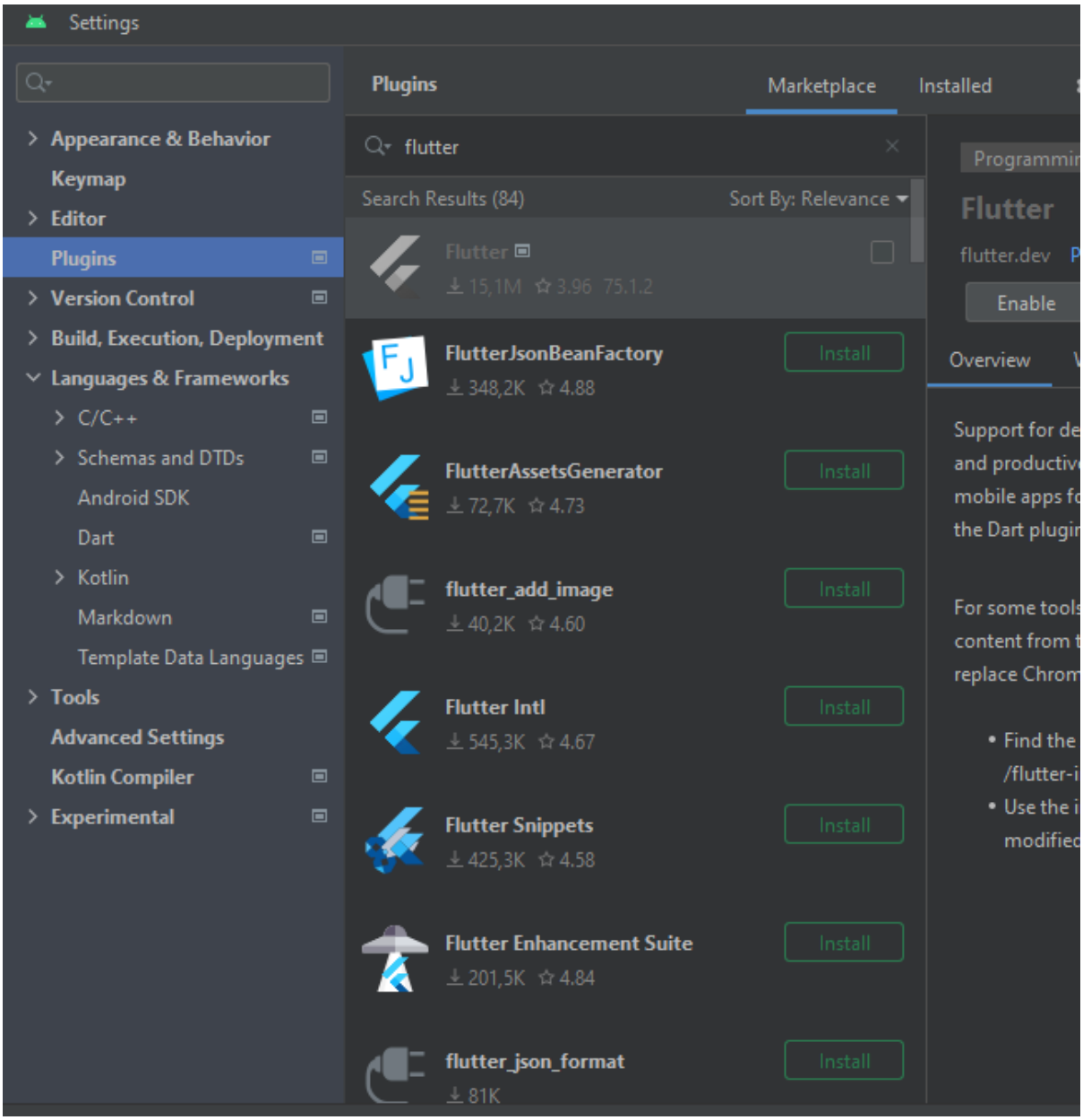
Selecione o diretório de instalação



4. Configure para o flutter

Assim que instalar, abra o aplicativo e acesse suas configurações.
Na aba “Plugins”, digite “Flutter” e instale


File
Edit
View
Navigate
Code
Refactor
Build
New
Open...
Profile or Debug APK
Recent Projects
Close Project
Settings... Ctrl+Alt+S
Project Structure... Ctrl+Alt+Shift+S
File Properties
Local History
Save All Ctrl+S
Reload All from Disk Ctrl+Alt+Y
Repair IDE
Invalidate Caches...
Manage IDE Settings
New Projects Setup
Export
Print...
Power Save Mode
Essential Highlighting
Exit
windows



CONCEITOS ESSENCIAIS

1. Widgets

Widgets são elementos básicos da interface gráfica que atuam como blocos de construção utilizados para criar a interface do usuário e adicionar ao sistema operacional uma tarefa em específico[3]. Exemplos incluem botões, caixas de texto, sliders, entre outros. Eles são fundamentais porque permitem a interação do usuário com o aplicativo.

[3] "O que é widget? - Canaltech." 4 ago.. 2021, https://canaltech.com.br/software/o-que-e-widget/. Acessado em 26 nov.. 2023.

Cada elemento na tela de um aplicativo Flutter é um widget, que pode ser subdividido ou envolvido em um outro maior.

2. Programação Orientada a Objetos

É um paradigma de programação baseado no conceito de "objetos". Estes objetos podem conter dados, na forma de campos, frequentemente conhecidos como atributos; e códigos, na forma de procedimentos, frequentemente conhecidos como métodos.

A POO, como é chamada, possui conceitos fundamentais como **encapsulamento, herança, polimorfismo e abstração**, necessários para que se desenvolva um código da melhor forma.

Um recurso chave da POO é a capacidade de reutilizar código e a facilidade de modificá-lo conforme necessário. Está presente em linguagens populares como Java, C++, C# e Python.

3. Classes

Classes são modelos a partir dos quais outros objetos são criados. Através dela são definidos atributos e métodos comuns a todos os objetos gerados, que herdam suas características

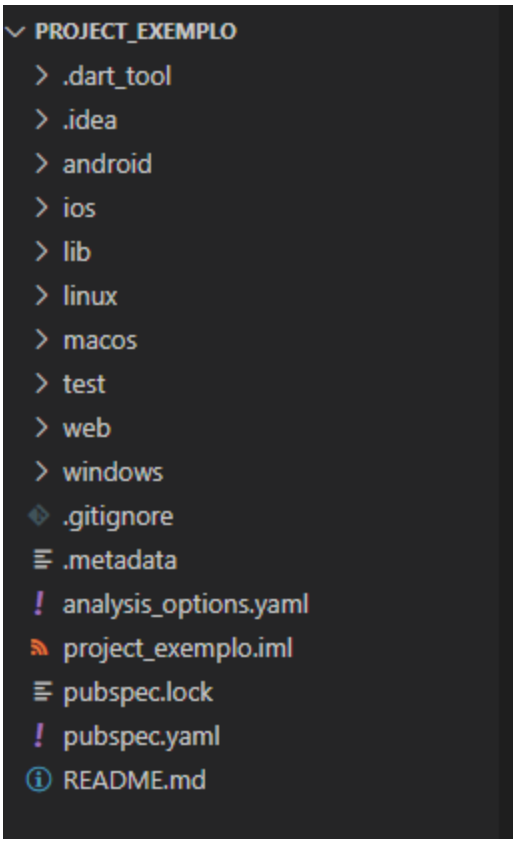
4. Firebase

É uma plataforma desenvolvida pelo Google para facilitar a criação de aplicativos móveis e da web. Ela oferece uma variedade de serviços e ferramentas que ajudam os desenvolvedores a construir, gerenciar e crescer seus aplicativos com facilidade. Isso inclui, mas não está limitado a, autenticação de usuários, armazenamento de dados, análise, notificações push, entre outros. É popular porque simplifica muitas das operações de backend e de infraestrutura que os desenvolvedores precisam.

METODOLOGIA - DESENVOLVIMENTO

Após a instalação do Flutter e a sua configuração no Visual Studio Code, é possível criar um nove projeto através do terminal ou do cmd. **Neste caso abaixo, foi criado um projeto chamado project_exemplo no diretório ApostilaTCC.**

```
PS C:\Users\User\Documents\Escola\PJS> cd ApostilaTCC
PS C:\Users\User\Documents\Escola\PJS\ApostilaTCC> flutter create project_exemplo
```



Automaticamente, ele cria pastas e documentos que contém o essencial para o projeto. Nem todas precisarão ser acessadas ou modificadas[4] , mas algumas são fundamentais, principalmente a **lib** e o arquivo **pubspec.yaml.**

Na pasta lib estarão contidos todos os arquivos .dart necessários para rodar o código, sendo automaticamente criado o **main.dart**.

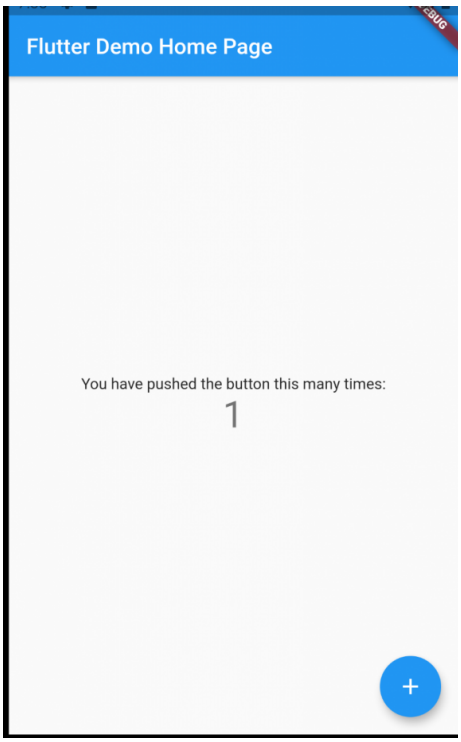
[4] Algumas pastas sequer é indicado que sejam acessadas a não ser que o programador tenha certeza do que está alterando, sendo este o caso das pastas referentes às plataformas, **android**, **ios**, etc.

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        //
        // TRY THIS: Try running your application with "flutter run". You'll see
```



Essa é a tela **default** feita pelo próprio Flutter, que apresenta conceitos básicos como os **widgets**, que serão fundamentais para o desenvolvimento do projeto.

A pubspec.yaml é um manifesto para a aplicação que contém uma série de informações importantes como:

- **Dependências:** Bibliotecas externas (packages) que o projeto necessita, incluindo os plugins que interagem com recursos do dispositivo como câmera, GPS, etc.

- **Versão do Projeto:** Define a versão atual do aplicativo
- **Configurações do Ambiente**: Especifica a versão do SDK do Dart que o aplicativo requer
- **Assets:** Declara os recursos que o aplicativo utilizará e que estão contidos na pasta “assets”

1. SplashScreen

O aplicativo se inicia com uma “SplashScreen”, enquanto carrega, que foi definida no arquivo carregamento.dart

```dart
@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: Colors.transparent,
    resizeToAvoidBottomInset: false,
    body: SingleChildScrollView(
      child: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Image.asset(
              'assets/loading.png',
              fit: BoxFit.cover, // Defina BoxFit.cover para pre
              width: MediaQuery.of(context).size.width, // Largu
              height: MediaQuery.of(context).size.height, // Alt
            ), // Image.asset
          ],
        ), // Column
      ), // Center
    ), // SingleChildScrollView
  ); // Scaffold
}
```

Essa é a tela de carregamento

2. Tela de Cadastro e Login

```dart
class _TelaCadastroState extends State<TelaCadastro> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  final TextEditingController _smartphoneController = TextEditingController();

  final List<String> _opcoesGenero = ['Masculino', 'Feminino', 'Outro'];
  bool _isLoading = false;
  bool _isChecked = false;
  String _emailError = '';
  String _senhaError = '';
  String _celularError = '';

  var formataCELULAR = MaskTextInputFormatter(
      mask: '(##) #####-####', filter: {"#": RegExp(r'[0-9]')}); // MaskTextInputFormatter

  Future<void> _signUp() async {
    setState(() {
      _isLoading = true;
    });

    try {
      UserCredential userCredential =
          await _auth.createUserWithEmailAndPassword(
        email: _emailController.text,
        password: _passwordController.text,
      );
      Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => UserProfilePage(userCredential.user!)), // MaterialPageRoute
      );
    } catch (error) {
      setState(() {
        _isLoading = false;
      });
    }
  }
```

O cadastro foi possível de ser feito através do Firebase Authentication, utilizando recursos para criar usuários e acessá-los no banco de dados. Essa imagem é uma pequena parte do código, que se estende por 830 linhas.

3. Tela Principal (Home)

```
void verificaLista(BuildContext context) {
  final doacoesProvider =
      Provider.of<DoacoesProvider>(context, listen: false);

  setState(() {
    if (doacoesProvider.listaDoacoes.isNotEmpty) {
      mostrarWidget = true;
      doacao = doacoesProvider.listaDoacoes[0];
    } else {
      mostrarWidget = false;
      doacao = DoacaoSelecionada(
        // Inicialize com um valor padrão se a lista estiver vazia
        cidadeSelecionada: 'Cidade Padrão',
        periodoSelecionado: 'Período Padrão',
        horarioSelecionado: 'Horário Padrão',
        diadaSemana: 'Dia padrão',
        key: 'Chave padrão',
        selectedDate: DateTime.now(),
      ); // DoacaoSelecionada
    }
  });
}

verificaCPF() {
  if (_userData?['cpf'] == '') {
    return 'indefinido';
  } else {
    return _userData?['cpf'] ?? 'erro';
  }
}
```

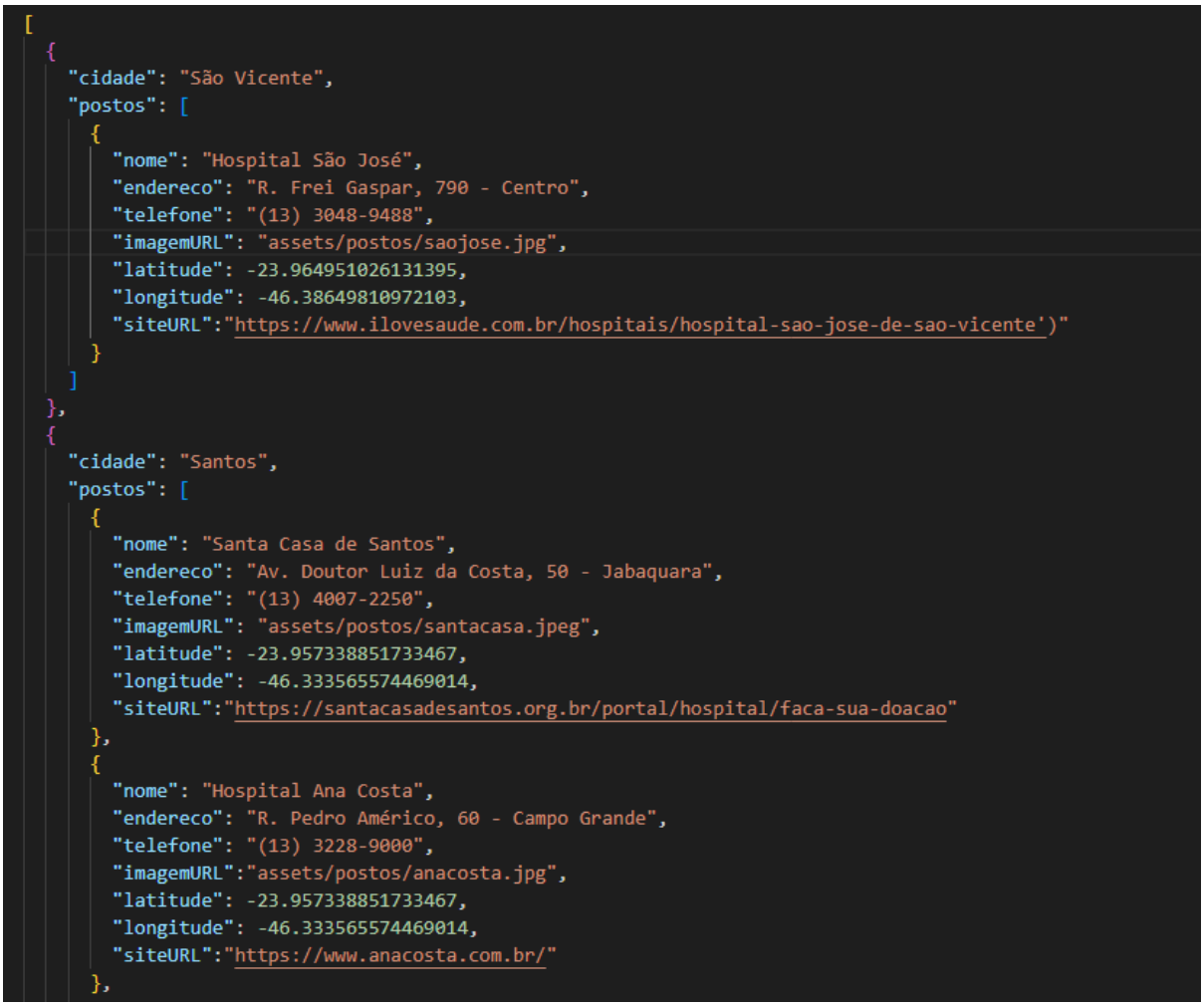
Através de funções assíncronas e do método **Provider**, que permite a reutilização de widgets em outras telas, foi possível atualizar a tela Home de acordo com as informações cadastradas do usuário

4. SideBar

```
backgroundColor: Colors.white,
//appBar: AppBar(
//title: Text('Minha Aplicação'),
// ),
drawer: Drawer(
  child: ListView(
    padding: EdgeInsets.zero,
    children: <Widget>[
      UserAccountsDrawerHeader(
        decoration: BoxDecoration(color: Theme.of(context).primaryColor),
        accountName: Text(_user?.displayName ?? 'Nome do Usuário'),
        accountEmail: Text(_user?.email ?? 'email@example.com'),
        currentAccountPicture: FutureBuilder<String>(
          future: getImageURL(),
          builder:
              (BuildContext context, AsyncSnapshot<String> snapshot) {
            if (snapshot.connectionState == ConnectionState.done &&
                snapshot.hasData) {
              final profileImageURL = snapshot.data;
              return CircleAvatar(
                radius: 30,
                backgroundColor: Colors.white,
                backgroundImage: NetworkImage(profileImageURL!),
              ); // CircleAvatar
            } else if (snapshot.hasError) {
              print('Erro ao obter URL da imagem: ${snapshot.error}');
              return CircleAvatar(
                radius: 30,
                backgroundColor: Colors.white,
                child: Icon(
                  Icons.person,
                  size: 30,
                  color: Colors.grey,
                ), // Icon
              ); // CircleAvatar
            } else {
              return CircularProgressIndicator();
            }
```

Através de um widget ListView foi construída a barra lateral contendo foto de perfil e informações do usuário, redirecionando às outras telas do aplicativo

5. Agendamento



```
[
  {
    "cidade": "São Vicente",
    "postos": [
      {
        "nome": "Hospital São José",
        "endereco": "R. Frei Gaspar, 790 - Centro",
        "telefone": "(13) 3048-9488",
        "imagemURL": "assets/postos/saojose.jpg",
        "latitude": -23.964951026131395,
        "longitude": -46.38649810972103,
        "siteURL":"https://www.ilovesaude.com.br/hospitais/hospital-sao-jose-de-sao-vicente')"
      }
    ]
  },
  {
    "cidade": "Santos",
    "postos": [
      {
        "nome": "Santa Casa de Santos",
        "endereco": "Av. Doutor Luiz da Costa, 50 - Jabaquara",
        "telefone": "(13) 4007-2250",
        "imagemURL": "assets/postos/santacasa.jpeg",
        "latitude": -23.957338851733467,
        "longitude": -46.333565574469014,
        "siteURL":"https://santacasadesantos.org.br/portal/hospital/faca-sua-doacao"
      },
      {
        "nome": "Hospital Ana Costa",
        "endereco": "R. Pedro Américo, 60 - Campo Grande",
        "telefone": "(13) 3228-9000",
        "imagemURL":"assets/postos/anacosta.jpg",
        "latitude": -23.957338851733467,
        "longitude": -46.333565574469014,
        "siteURL":"https://www.anacosta.com.br/"
      },
```

Para o agendamento foi necessário um arquivo **JSON** com as informações dos postos de doação que é acessado através de uma função assíncrona. O arquivo se localiza na pasta assets, dentro do projeto

```dart
@override
void initState() {
  super.initState();
  // Chamada para carregar os postos de doação
  carregarPostos().then((postosCarregados) {
    // Atualize o estado da aplicação com os postos carregados
    setState(() {
      cidades = postosCarregados;
    });
  });
}

Future<List<CidadeSelecionada>> carregarPostos() async {
  String data = await rootBundle.loadString('assets/json/postos.json');
  List<dynamic> json = jsonDecode(data);

  return json.map((item) => CidadeSelecionada.fromJSON(item)).toList();
}

void onDropdownChanged(String newValue) {
  setState(() {
    dropdownValueCidade = newValue;
    dropdownValuePeriodo = 'Manhã';
    dropdownValueHorario = horariosPorPeriodo['Manhã']!.first;
    deselectedAllPostos(); // Desselecionar todos os postos ao trocar de cidade
    selectedPosto = null;

    final healthCentersInSelectedCity =
        healthCenters.where((center) => center.cidade == newValue).toList();
```

6. MargaRED (Chatbot)

O chatbot da Margared foi feito utilizando várias funções e widgets como DropdownButton e DropdownMenuItem para construir um chat dinâmico ainda que com respostas e perguntas pré-definidas, as quais estão contidas em um arquivo JSON também.

```
Padding(
  padding: const EdgeInsets.all(8.0),
  child: Row(
    children: <Widget>[
      Expanded(
        child: Theme(
            data: Theme.of(context)
                .copyWith(canvasColor: const Color(0xFFB63006)),
            child: Container(
              padding: const EdgeInsets.all(3),
              decoration: BoxDecoration(
                color: Colors.white,
                borderRadius: BorderRadius.circular(10),
              ), // BoxDecoration
              child: DropdownButton<String>(
                isExpanded: true,
                icon: const Icon(Icons.wechat),
                value: selectedQuestion,
                // Estilo quando o DropdownButton NÃO está selecionado
                style: const TextStyle(
                  color: Colors
                      .white, // Cor do texto quando não está selecionado
                  fontSize: 15.0,
                  fontFamily: 'GlacialIndifference',
                ), // TextStyle
                // Estilo quando o DropdownButton ESTÁ selecionado
                selectedItemBuilder: (BuildContext context) {
                  return predefinedQuestions.map((String question) {
                    return DropdownMenuItem<String>(
                      value: question,
                      child: Text(
                        question,
                        style: const TextStyle(
                          color: Colors
                              .black, // Cor do texto quando selecionado
                          fontSize: 15.0,
                          fontFamily: 'GlacialIndifference',
                        ), // TextStyle
                      ), // Text
                    ); // DropdownMenuItem
```

7. Mapa

O mapa foi desenvolvido através de uma **API**[5] do Google Maps, disponibilizada pelo Flutter e que permite integrar os recursos de GPS com o aplicativo

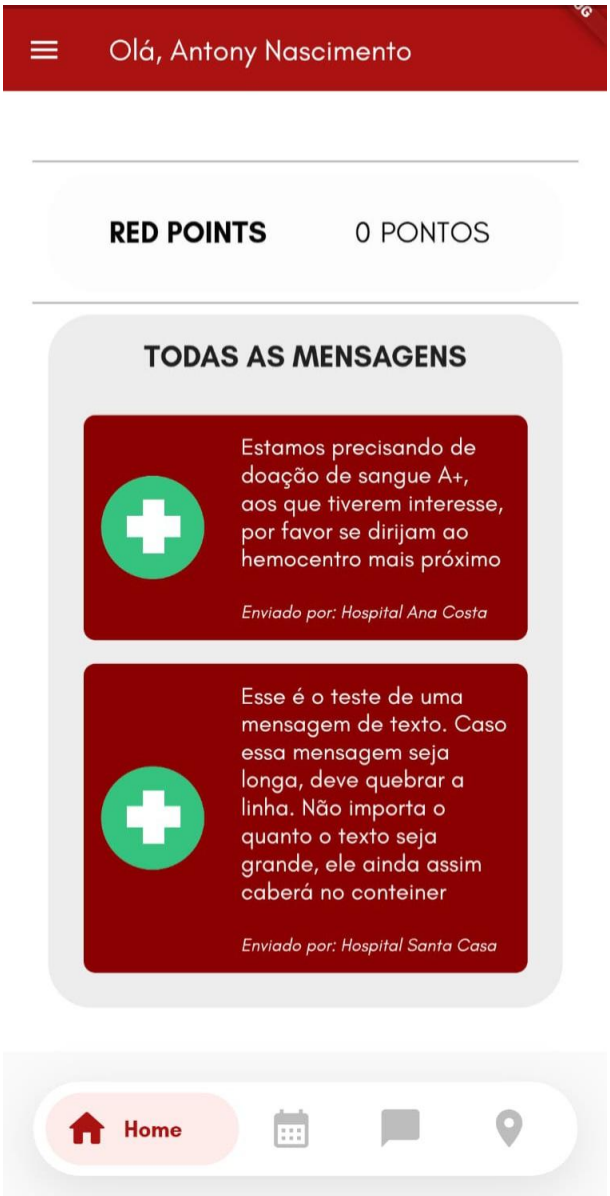
[5] API, sigla para "Interface de Programação de Aplicações" (em inglês, "Application Programming Interface"), é um conjunto de regras e definições que permite que softwares diferentes se comuniquem entre si.

```
), // AppBar
body: Column(
  children: [
    SizedBox(
      height: 400,
      child: GoogleMap(
        onMapCreated: (controller) {
          setState(() {
            _mapController = controller;
          });
        },
        initialCameraPosition: CameraPosition(
          target: LatLng(-23.9535, -46.3961),
          zoom: 12.0,
        ), // CameraPosition
        markers: healthCenters
            .map((center) => Marker(
                  markerId: MarkerId(center.nome),
                  position: LatLng(center.latitude, center.longitude),
                  infoWindow: InfoWindow(
                    title: center.nome,
                    snippet: '${center.endereco}\n${center.telefone}',
                  ), // InfoWindow
                  icon: BitmapDescriptor.defaultMarkerWithHue(
                    BitmapDescriptor.hueRed,
                  ),
                )) // Marker
            .toSet(),
      ), // GoogleMap
    ), // SizedBox
    Expanded(
      child: ListView.builder(
        itemCount: healthCenters.length,
        itemBuilder: (context, index) {
          final center = healthCenters[index];
          final isSelected = index == selectedCenterIndex;
```

APRESENTAÇÃO DAS TELAS

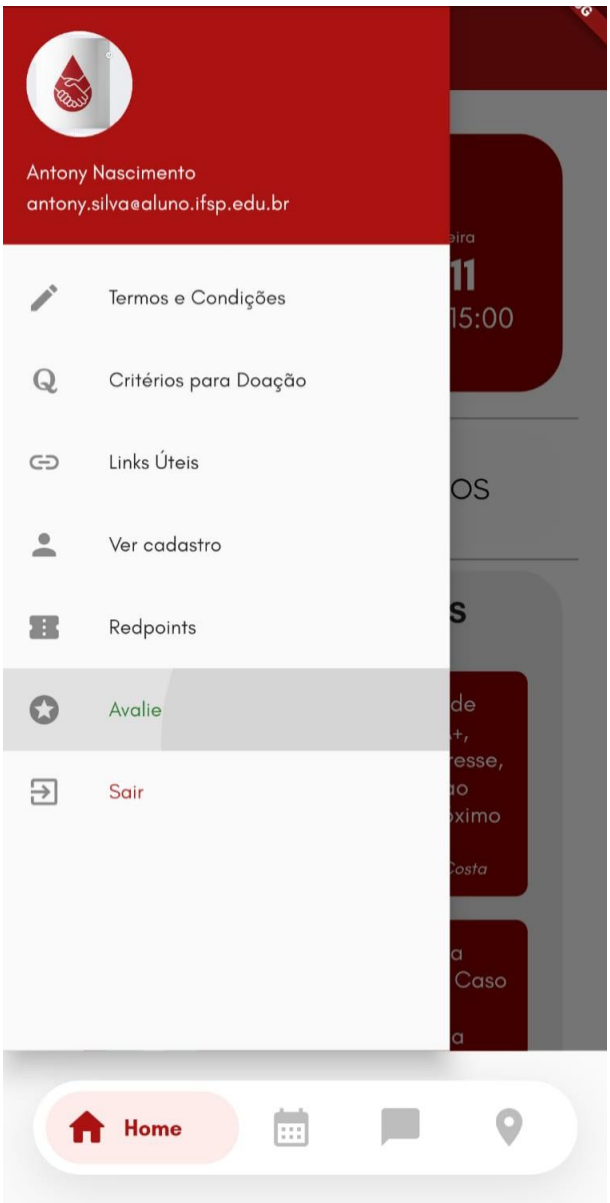
1. Página Inicial

No layout inicial do aplicativo, é apresentado um conjunto de elementos visuais que têm como objetivo facilitar a visualização da próxima doação agendada pelo usuário. Além disso, são exibidos os RedPoints acumulados, um sistema de pontos internos, e há um espaço dedicado para mensagens provenientes de hospitais. Estas mensagens descrevem casos específicos de pacientes que requerem doações de sangue urgentes.



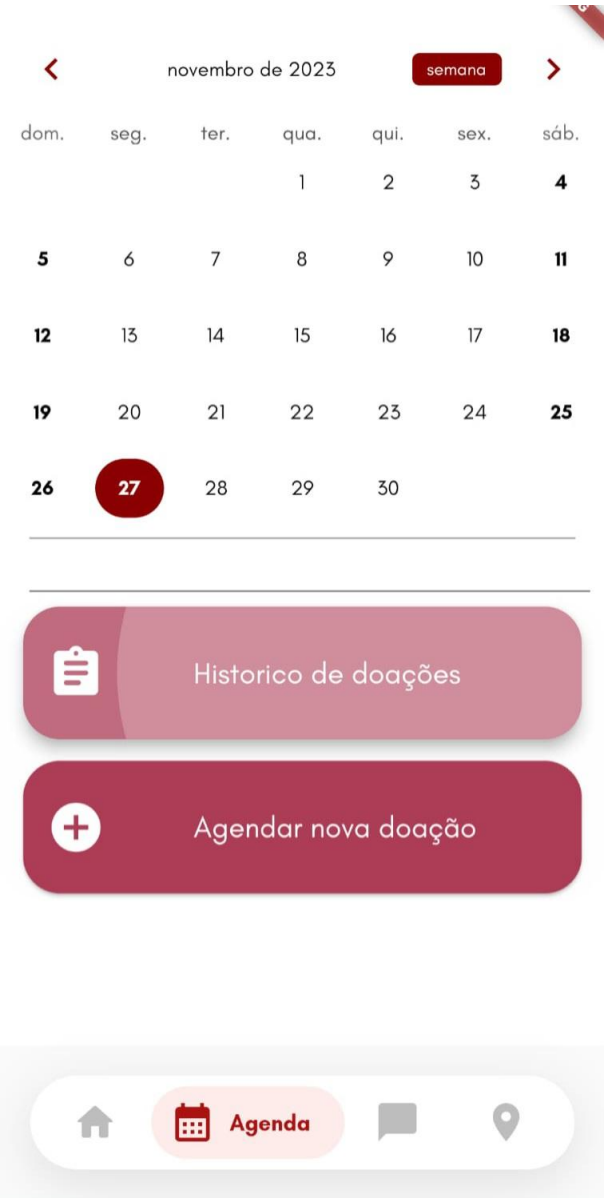
2. Aba Lateral

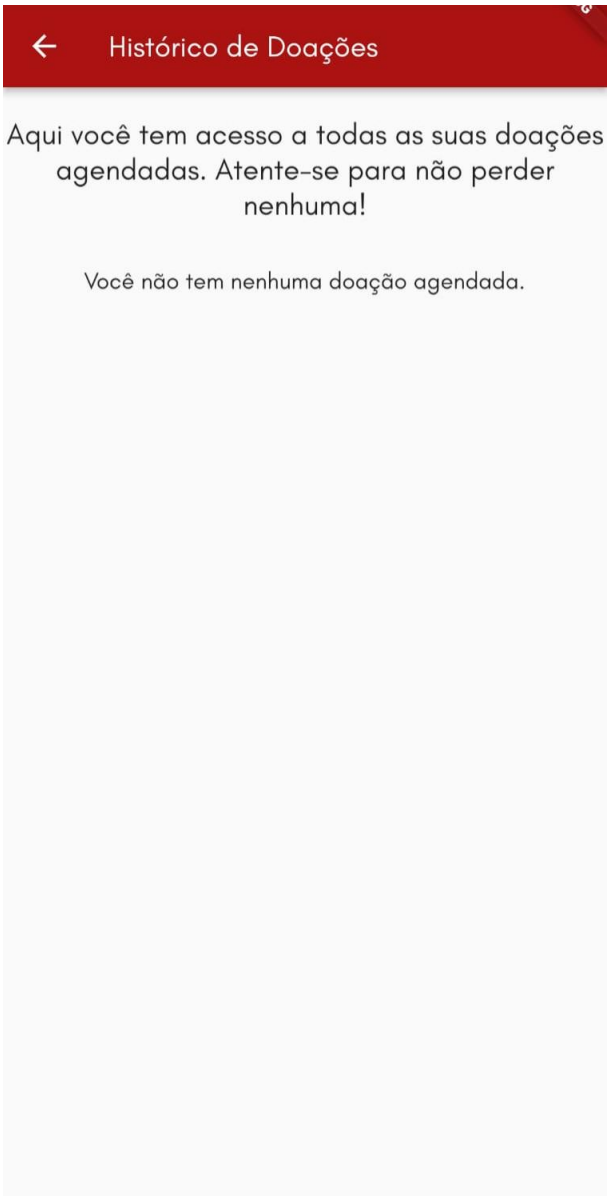
A seção da aba lateral oferece ao usuário a possibilidade de acessar diferentes seções do aplicativo, incluindo os termos e condições de uso, os critérios necessários para realizar doações, a área para avaliação do software e a opção de sair do aplicativo.



3. Agenda

Dentro da seção da Agenda, os usuários têm acesso a um calendário atualizado, facilitando a visualização da próxima doação agendada. Além disso, são disponibilizados botões para acessar o histórico de doações já realizadas e para agendar novas doações.


novembro de 2023
semana
dom. seg. ter. qua. qui. sex. sáb.
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
Historico de doações
Agendar nova doação
Agenda



4. Triagem

Após solicitar uma nova doação, os usuários passam por uma fase de triagem. Nessa etapa, são submetidos a uma série de perguntas específicas que visam determinar se estão aptos ou não para realizar a doação de sangue.

TRIAGEM PRÉVIA

A honestidade nessa etapa da doação de sangue é fundamental para preservar a saúde do doador e do paciente. Identifique as situações que estão sendo questionadas e responda com precisão cada questão. A pré-triagem eletrônica tem como objetivo evitar um deslocamento do candidato ao posto de coleta caso esteja impedido de doar sangue por um dos requisitos básicos.
Caso aprovado, a agenda online ficará aberta para escolha do melhor local, dia e horário; caso tenha algum impedimento, o sistema apresentará o tempo de inaptidão e abrirá a agenda após a data liberada para realizar a doação de sangue, dentro do prazo definido de abertura da agenda.

Contamos com seu apoio e boa doação!

Obs: Estas questões não esgotam os motivos de impedimentos para doação, de forma que outras informações prestadas por você durante a Triagem Clínica (no posto de coleta) também serão consideradas para definir se está apto para doar sangue nesse momento.

DADOS DO DOADOR

Nome Completo: Antony Nascimento
CPF: 666.666.666-66
RG: 66.666.666-6
Data de Nascimento: 66/66/6666
Sexo: Masculino
Nome da Mãe: Mãe
CEP: 11540-220
Endereço: Caminho São Geraldo, 160

TRIAGEM PRÉVIA

Nome da Mãe: Mãe
CEP: 11540-220
Endereço: Caminho São Geraldo, 160
Tipo de Sangue: O+

Doou sangue há menos de 2 meses?

- [ ] Não
- [ ] Sim

Fez implante dentário e concluiu o tratamento há menos de 7 dias?

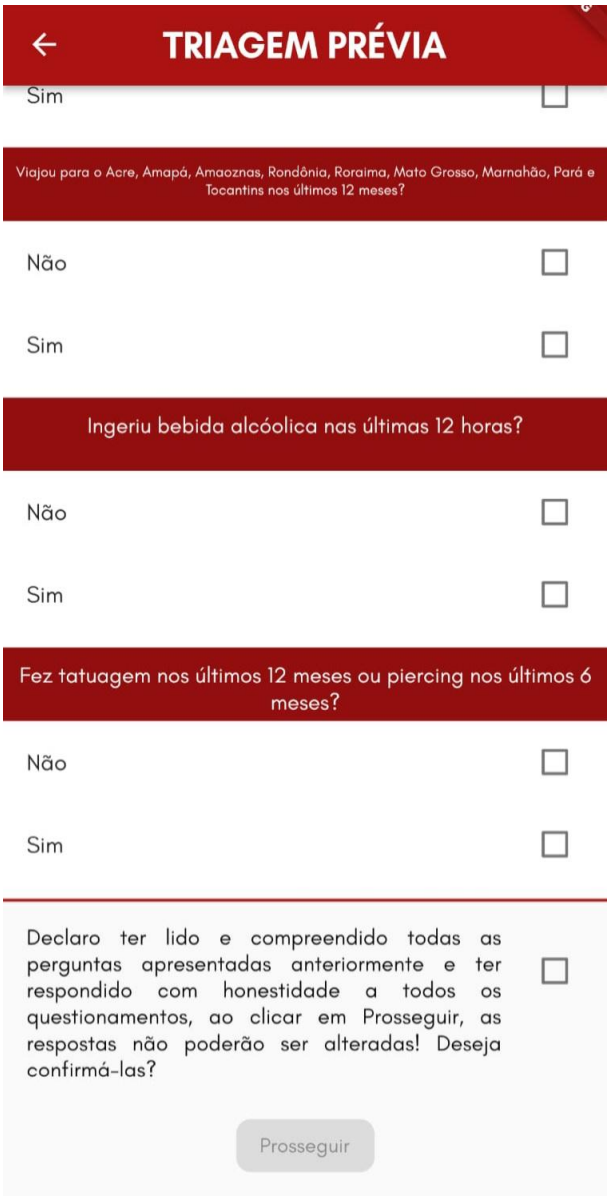
- [ ] Não
- [ ] Sim

Fez cirurgia de pequeno porte há menos de 3 meses?

- [ ] Não
- [ ] Sim

Fez exame / procedimento endoscópico há menos de 6 meses?

- [ ] Não
- [ ] Sim



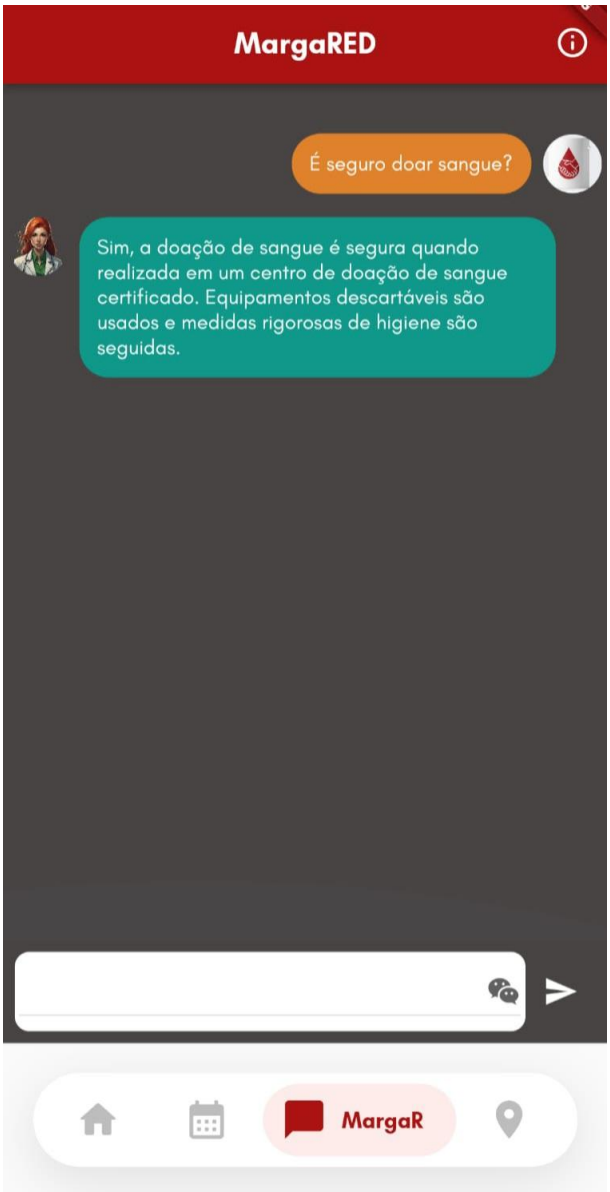
5. Agendamento

Para agendar uma nova doação, o usuário é guiado por um processo no qual seleciona, primeiramente, a cidade onde pretende realizar a doação. Em seguida, são solicitadas informações sobre o período do dia, a data, o horário desejado e, por último, mas igualmente importante, o posto de doação escolhido.


Agendamento
Selecione a cidade, período e horário:
São Vicente
Manhã
8:00 - 9:00
27/11/2023
Hospital São José
R. Frei Gaspar, 790 - Centro
Próximo

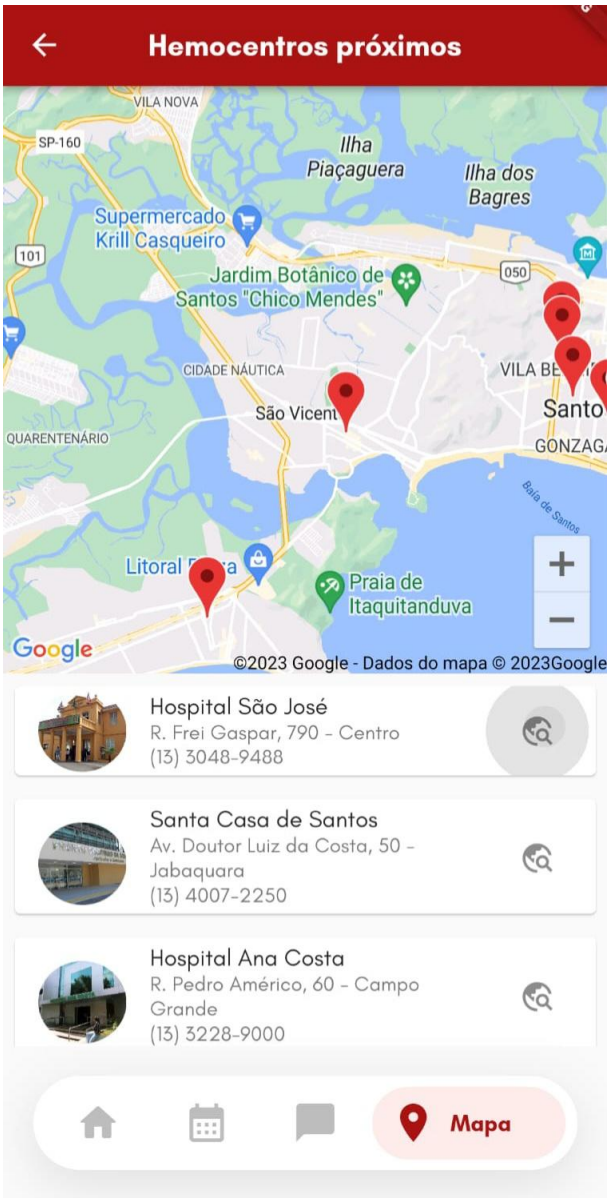
6. Margared

Margared desempenha o papel de assistente virtual dentro do aplicativo. Funciona como um sistema de perguntas frequentes (FAQ), respondendo a questões pré-selecionadas e fornecendo informações relevantes para os usuários.



7. Localização

Esta seção se utiliza da funcionalidade do Google Maps para oferecer aos usuários a visualização da localização geográfica de todos os postos de doação disponíveis, facilitando assim o acesso e a localização física dos locais para doação de sangue.



CONCLUSÃO

O projeto envolveu uma série de aprendizados adquiridos ao longo do curso e possibilitou a exploração de outras áreas até então desconhecidas para o grupo. Através desta apostila, não apenas é detalhado o processo de desenvolvimento como também possibilita que outros estudantes aprendam com os procedimentos realizados.