**Tutorial Flutter: Integração com APIs**

* **Recursos necessários:**
  + - Ambiente para desenvolvimento Flutter devidamente configurado
    - Emulador ou Dispositivo Android
    - Node.JS para criação da API
    - Visual Studio Code (Extensões Flutter)
* **Sinopse:**

Neste tutorial iremos realizar a criação de uma API REST própria e integração com Aplicação Flutter.

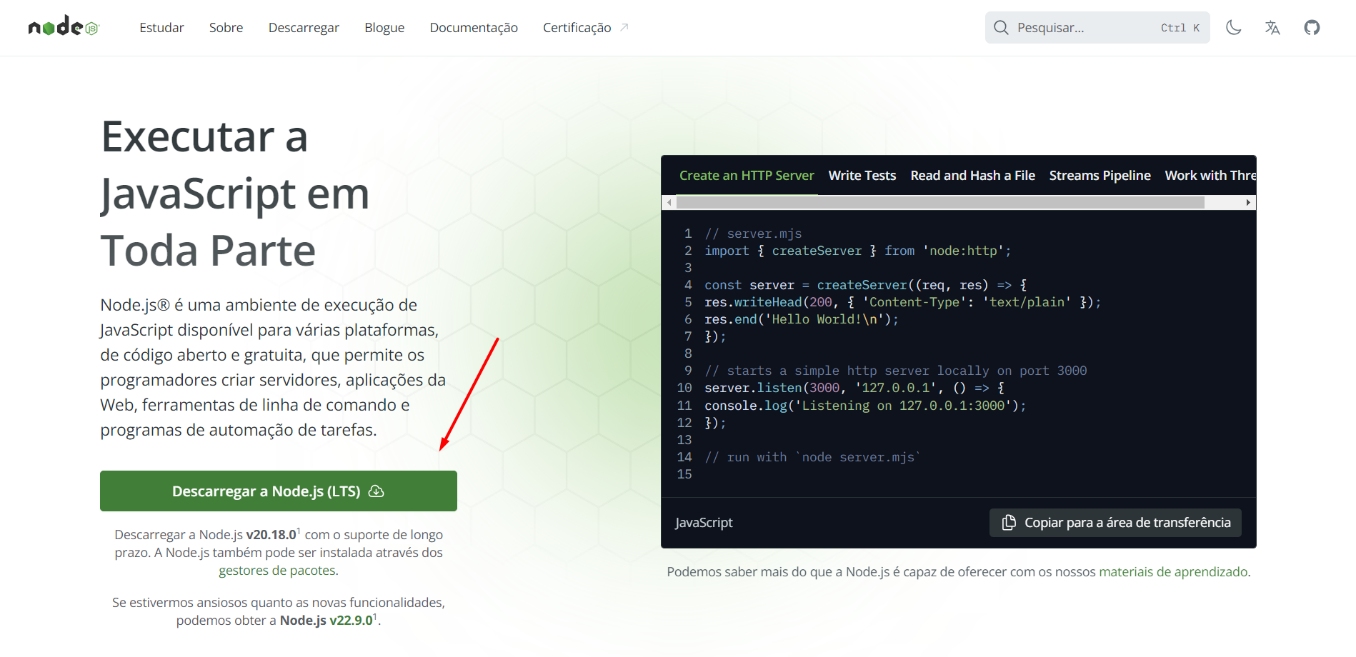
* **1. O que são APIs ?**

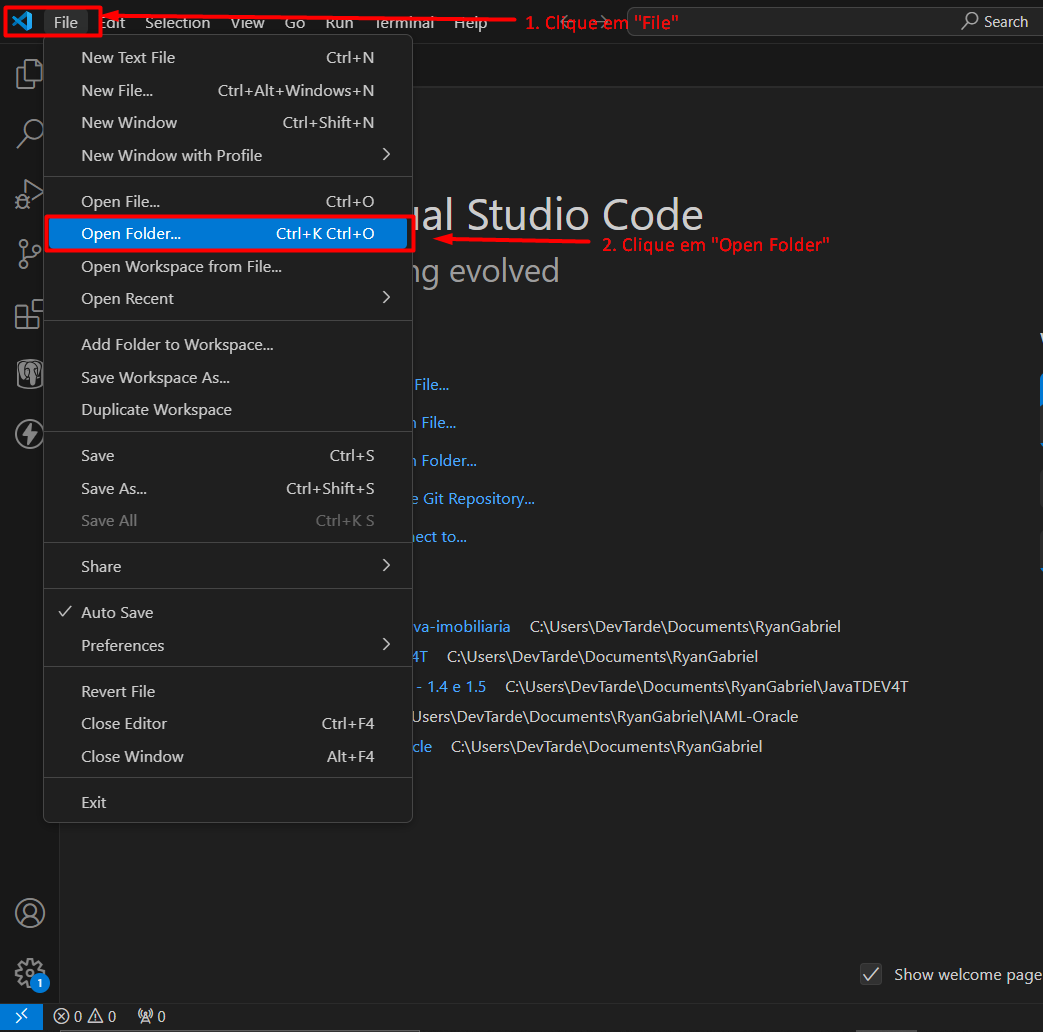
APIs (Application Programming Interfaces) são como pontes que permitem que diferentes sistemas, aplicativos ou serviços se comuniquem entre si. Imagine que você está em um restaurante: o garçom é a "API". Você (o cliente) não precisa ir até a cozinha para preparar a comida; você só faz o pedido ao garçom, e ele leva sua solicitação para a cozinha (o sistema) e, depois, traz de volta a comida pronta (a resposta).

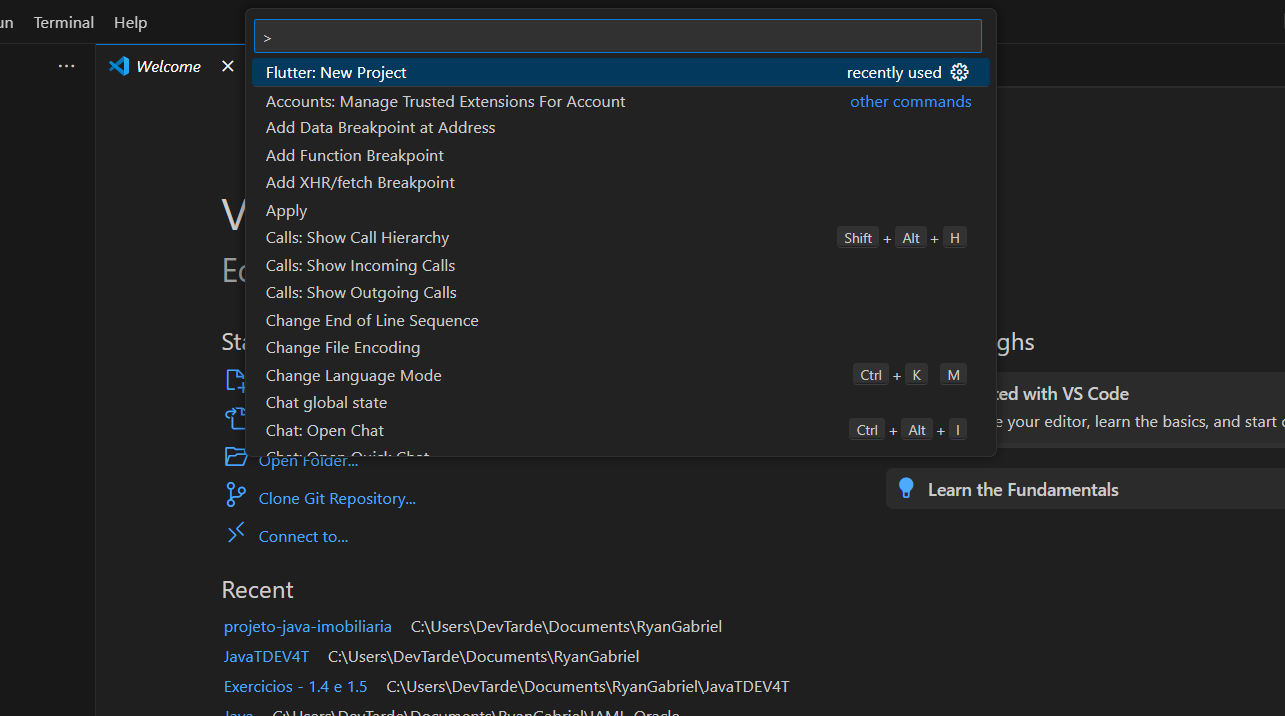
Quando você usa um app, ele pode precisar buscar informações de outro lugar, como dados de previsão do tempo ou os posts mais recentes do Instagram. Em vez de o app fazer tudo por conta própria, ele faz um "pedido" para a API, que se comunica com o servidor e traz a resposta pronta.

As APIs estão em, praticamente,

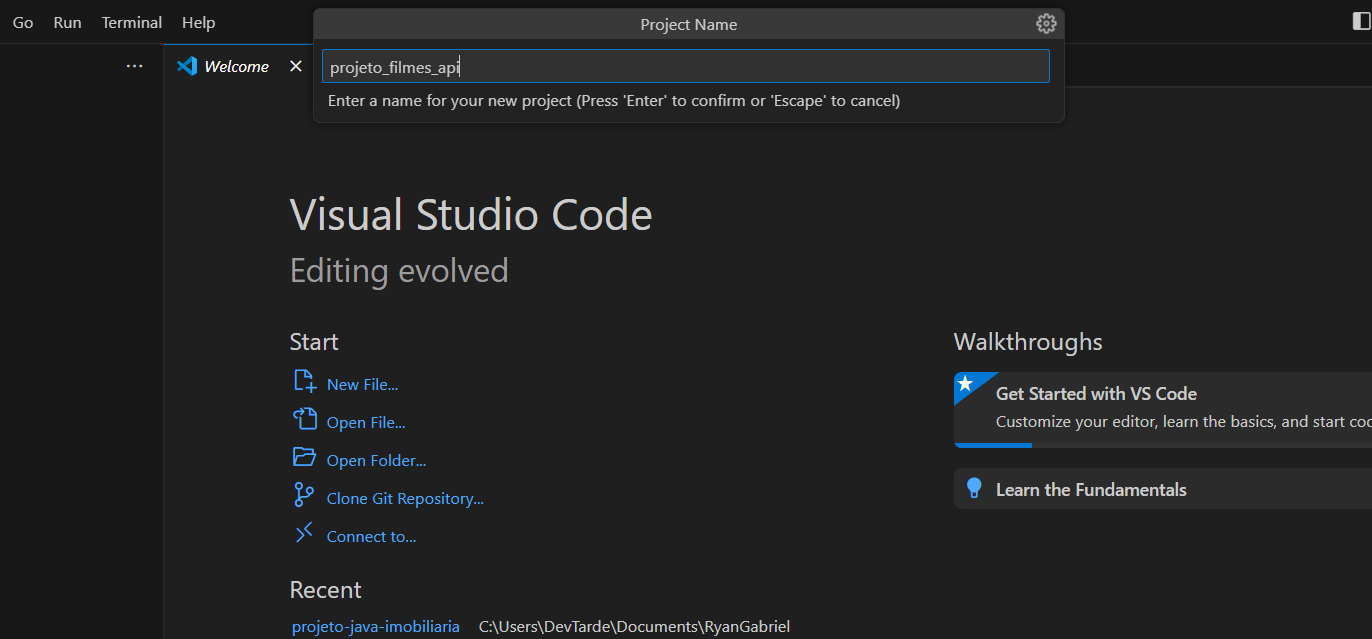
* + **Benefícios das APIs:**
    - **Facilidade de Integração:**  
      APIs facilitam a conexão entre diferentes sistemas e serviços, permitindo que eles troquem dados sem complicação. Você pode usar APIs para conectar seu app a serviços como pagamentos ou dados externos, sem precisar construir tudo do zero.
    - **Reutilização de Funcionalidades:**  
      Com uma API, você pode usar funções já prontas e testadas por outras empresas ou desenvolvedores. Um exemplo clássico é usar a API do Google Maps para mostrar mapas no seu app.
    - **Escalabilidade:**  
      APIs ajudam seu sistema a crescer sem perder eficiência. Elas permitem que partes diferentes do sistema trabalhem juntas de forma eficiente, tornando mais fácil adicionar novas funcionalidades ou suportar mais usuários.
    - **Segurança:**  
      APIs oferecem controle sobre quem pode acessar seus dados, exigindo autenticação e verificações, o que garante que apenas pessoas autorizadas possam usar certos recursos.
    - **Padronização:**  
      As APIs seguem regras e formatos bem definidos, como REST ou GraphQL, facilitando o uso e a integração entre desenvolvedores e sistemas, pois todos sabem como as coisas devem funcionar.
    - **Economia de Tempo e Recursos:**  
      Usar APIs economiza tempo e dinheiro, porque você aproveita infraestrutura e funcionalidades já existentes em vez de gastar para criar tudo do zero.
    - **Manutenção Fácil:**  
      APIs organizam e isolam as funcionalidades, o que facilita a atualização e manutenção do código sem quebrar outras partes do sistema.
* 2. Criando uma API:
  + 2.1 – Instalando o NODE:
    - Link: <https://nodejs.org/pt> ;
    - Clique no botão principal “ Descarregar a Node.js (LTS) “;

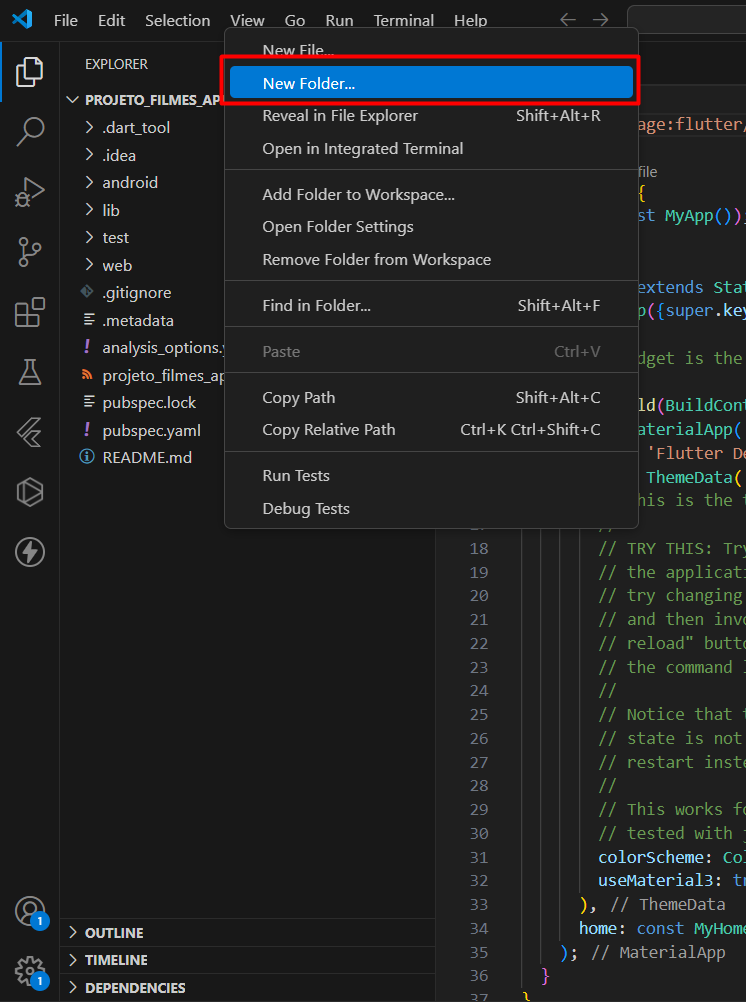


* Após baixar o instalador, execute-o e apenas prossiga com a instalação;
* **2.2 – Criando um projeto Flutter no VSCode:**
  + Abra uma nova pasta no seu VSCode;
  + Selecione o seu diretório preferido e abra, assim que aberto pressione sucessivamente “CTRL+SHIFT+P”, e procure pela opção: ‘ Flutter: New Project ‘

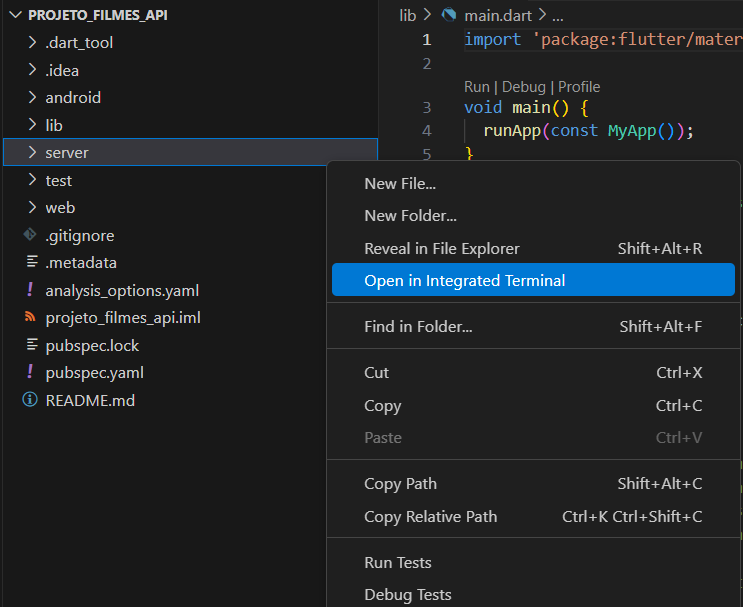


* + Pressione ‘ENTER’ e depois escolha a opção: ‘Application’;
  + Selecione o mesmo diretório que você está e depois coloque o nome do projeto com projeto\_filmes\_api;



* + Pressione ‘ENTER’ e aguarde o projeto ser criado;
  + Assim que criado crie uma nova pasta na raiz do projeto chamada ‘server’;

* + Clique com o botão direito na pasta server e abra um terminal integrado a ela:



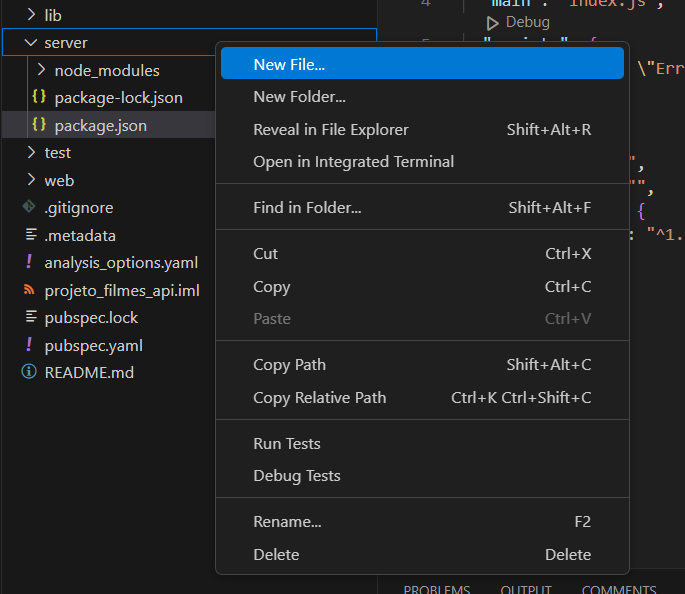
* + No terminal digite:

npm init -y

* + Aperte ENTER e aguarde um instante, depois digite novamente no terminal:

npm install json-server

* + Pressione ENTER novamente e aguarde a instalação..
  + Depois de instalado verifique se foi criado 2 arquivos JSON e um diretório (node\_modules);
  + Agora, dentro da pasta ‘server’, crie o arquivo “db.json”:



* + Dentro do arquivo “db.json” escreva a seguinte coleção:

{

    "filmes": [

      {

        "id": "cd68",

        "nome": "Os Vingadores",

        "categoria": "Acao",

        "duracao": 143,

        "ano": 2012,

        "classificacao": 9

      },

      {

        "id": "bbec",

        "nome": "Harry Potter e a Pedra Filosofal",

        "categoria": "Fantasia",

        "duracao": 152,

        "ano": 2001,

        "classificacao": 10

      },

      {

        "id": "ce65",

        "nome": "Esposa de Mentirinha",

        "categoria": "Comedia/Romance",

        "duracao": 117,

        "ano": 2011,

        "classificacao": 12

      },

      {

        "id": "b0e7",

        "nome": "Gente Grande 2",

        "categoria": "Comedia",

        "duracao": 101,

        "ano": 2013,

        "classificacao": 11

      },

      {

        "id": "5c6c",

        "nome": "Bad Boys IV",

        "categoria": "Ação",

        "duracao": 150,

        "ano": 2024,

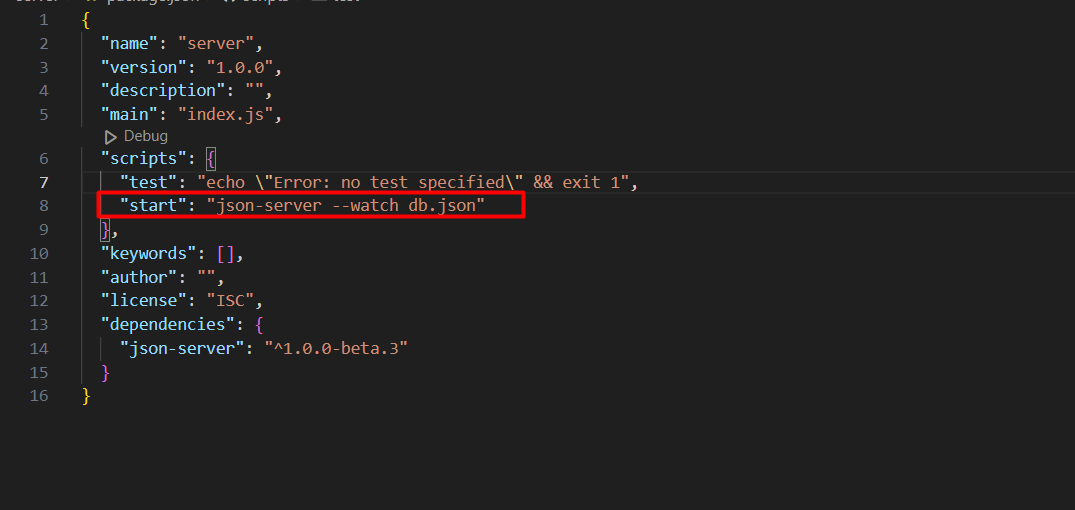
        "classificacao": 10

      }

    ]

  }

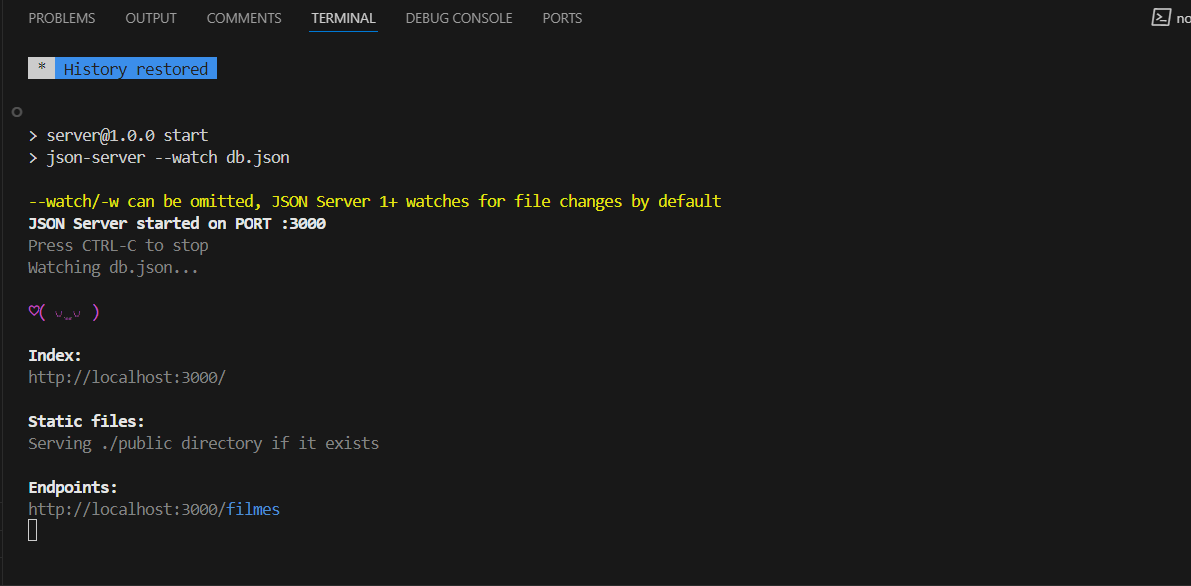
* + Agora procure pelo arquivo “package.json”, e adicione a linha:



* Depois salve e escreva no terminal:

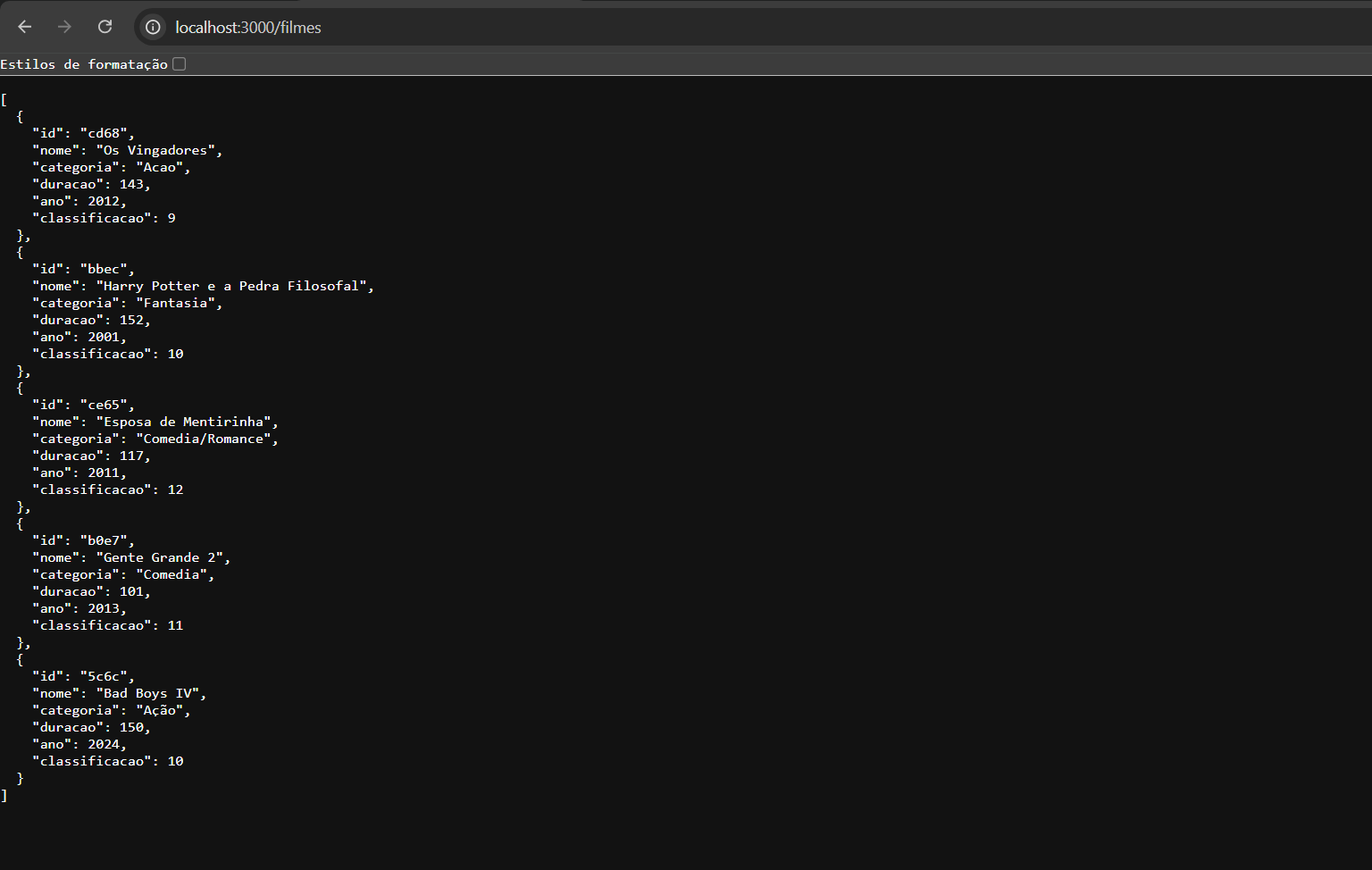
npm start

* Se tudo der certo esse serão resultado:



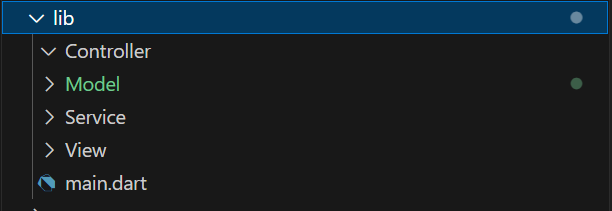
Isso significa que a API está rodando em nossa máquina na porta 3000 com o endereço: /filmes, podemos verificar a API pelo próprio navegador. Experimente digitar no seu navegador o endereço: localhost:3000/filmes.

* Resultado esperado:



Agora que temos a nossa API feita podemos começar a criar nosso aplicação em Flutter:

* Vá até a pasta ‘lib’ e crie os seguintes diretórios:



* Começando pelo model, crie uma nova classe Filme.dart e coloque o seguinte código:

class Filme {

  // Atributos

  final String? id; // Permite que o id seja nulo

  final String nome;

  final String categoria;

  final int duracao;

  final int ano;

  final int classificacao;

  // Construtor

  Filme(

      {this.id, // Id não obrigatório

      required this.nome,

      required this.categoria,

      required this.duracao,

      required this.ano,

      required this.classificacao});

  // Fábrica (factory) para criar uma instância de Filme a partir de um JSON

  factory Filme.fromJson(Map<String, dynamic> json) {

    return Filme(

      id: json['id'], // Converte o valor do campo 'id' do JSON para o atributo 'id'

      nome: json['nome'], // Converte o valor do campo 'nome' do JSON para o atributo 'nome'

      categoria: json['categoria'], // Converte o valor do campo 'categoria' do JSON para o atributo 'categoria'

      duracao: json['duracao'], // Converte o valor do campo 'duracao' do JSON para o atributo 'duracao'

      ano: json['ano'], // Converte o valor do campo 'ano' do JSON para o atributo 'ano'

      classificacao: json['classificacao'],// Converte o valor do campo 'classificacao' do JSON para o atributo 'classificacao'

    );

  }

  // Método para converter uma instância de Filme em um Map (JSON)

  Map<String, dynamic> toJson() {

    return {

      'nome': nome, // Converte o atributo 'nome' para o campo 'nome' do JSON

      'categoria': categoria, // Converte o atributo 'categoria' para o campo 'categoria' do JSON

      'duracao': duracao, // Converte o atributo 'duracao' para o campo 'duracao' do JSON

      'ano': ano, // Converte o atributo 'ano' para o campo 'ano' do JSON

      'classificacao': classificacao, // Converte o atributo 'classificacao' para o campo 'classificacao' do JSON

    };

  }

}

* Agora vamos para a pasta ‘Service’, crie a classe FilmeService.dart e insira o seguinte código:
* Na linha ‘http://SEU\_IP:3000/filmes’, substitua o SEU\_IP pelo ipv4 da sua máquina, para pegar basta abrir o CMD do windows e digitar ipconfig.

import 'dart:convert'; // Necessário para converter dados JSON

import 'package:http/http.dart' as http; // Biblioteca HTTP para realizar requisições

import 'package:projeto\_filmes\_api/Model/Filme.dart'; // Importa o modelo de Filme

class FilmeService {

  // URL base da API

  final String baseUrl = 'http://SEU\_IP:3000/filmes';

  // Método para buscar a lista de filmes

  Future<List<Filme>> getFilmes() async {

    final url = Uri.parse(baseUrl); // Constrói a URL

    final response = await http.get(url); // Requisição GET para buscar filmes

    if (response.statusCode == 200) {

      // Se a requisição foi bem-sucedida

      List<dynamic> jsonList = jsonDecode(response.body); // Decodifica o JSON

      return jsonList.map((json) => Filme.fromJson(json)).toList(); // Mapeia a lista de JSON para objetos Filme

    } else {

      throw Exception('Failed to load filmes'); // Exceção em caso de erro

    }

  }

  // Método para criar um novo filme

  Future<Filme> postFilme(Filme filme) async {

    final url = Uri.parse(baseUrl); // Constrói a URL

    final response = await http.post(

      url,

      body: jsonEncode(filme.toJson()), // Converte o objeto Filme para JSON

      headers: {'Content-Type': 'application/json'}, // Cabeçalhos da requisição

    );

    if (response.statusCode == 201) {

      // Se o filme foi criado com sucesso

      return Filme.fromJson(jsonDecode(response.body)); // Retorna o novo filme criado

    } else {

      throw Exception('Failed to create filme'); // Exceção em caso de erro

    }

  }

  // Método para deletar um filme por ID

  Future<void> deleteFilme(String id) async {

    final url = Uri.parse('$baseUrl/$id'); // Constrói a URL com o ID do filme

    final response = await http.delete(url); // Requisição DELETE

    if (response.statusCode != 200) {

      throw Exception('Failed to delete filme'); // Exceção em caso de erro

    }

  }

  // Método para atualizar um filme

  Future<Filme> putFilme(Filme filme) async {

    final url = Uri.parse(baseUrl); // Constrói a URL

    final response = await http.put(

      url,

      body: jsonEncode(filme.toJson()), // Converte o objeto Filme para JSON

      headers: {'Content-Type': 'application/json'}, // Cabeçalhos da requisição

    );

    if (response.statusCode == 200) {

      // Se a atualização foi bem-sucedida

      return Filme.fromJson(jsonDecode(response.body)); // Retorna o filme atualizado

    } else {

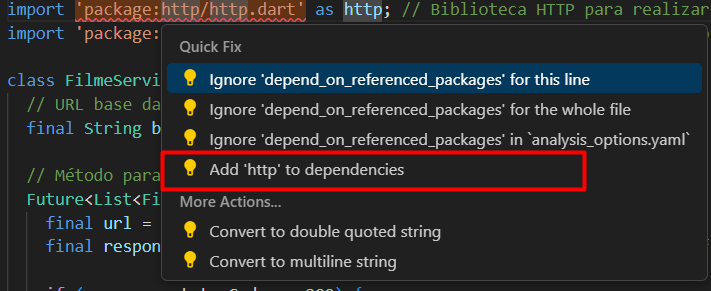
      throw Exception('Failed to update filme'); // Exceção em caso de erro

    }

  }

}

* Caso a linha abaixo fique com erro evidente, selecione ela e clique “Ctrl+.” selecione: *Add ‘http’ to dependencies* e aguarde.



* A classe FilmeService é a principal classe que faz as requisições diretamente a API, recebe os dados, converte-os para Dart e retorna as outras classes;
* Agora vamos seguir para o diretório Controller, crie a classe FilmeController.dart e insira o código:

import 'package:projeto\_filmes\_api/Model/Filme.dart'; // Importa o modelo de Filme

import 'package:projeto\_filmes\_api/Service/FilmeService.dart'; // Importa o serviço que faz a comunicação com a API

class FilmeController {

  List<Filme> listFilmes = []; // Lista que armazena os filmes recuperados da API

  final FilmeService \_service = FilmeService(); // Instância do serviço de filme

  // Método para buscar filmes do serviço

  Future<void> getFromFilmeFromService() async {

    try {

      listFilmes.clear(); // Limpa a lista antes de adicionar novos filmes

      listFilmes = await \_service.getFilmes(); // Chama o serviço para buscar filmes

      print('List filmes: $listFilmes'); // Log para ver a lista de filmes

    } catch (e) {

      print(e); // Log para ver o erro em caso de falha

    }

  }

  // Método para adicionar um filme

  Future<void> addFilme(Filme filme) async {

    try {

      Filme newFilme = await \_service.postFilme(filme); // Chama o serviço para adicionar o filme

      listFilmes.add(newFilme); // Adiciona o novo filme na lista

      print('Filme adicionado: $newFilme'); // Log para verificar o filme adicionado

    } catch (e) {

      print(e); // Log para capturar o erro

    }

  }

  // Método para deletar um filme

  Future<void> deleteFilme(String id) async {

    try {

      await \_service.deleteFilme(id); // Chama o serviço para deletar o filme

      listFilmes.removeWhere((filme) => filme.id == id); // Remove o filme da lista localmente

      print('Filme excluido: $id'); // Log para verificar o filme excluído

    } catch (e) {

      print(e); // Log para capturar o erro

    }

  }

  // Método para editar um filme

  Future<void> editFilme(Filme filme) async {

    try {

      await \_service.putFilme(filme); // Chama o serviço para atualizar o filme

      listFilmes.removeWhere((f) => f.id == filme.id); // Remove o filme antigo da lista

      listFilmes.add(filme); // Adiciona o filme atualizado na lista

      print('Filme editado: $filme'); // Log para verificar o filme editado

    } catch (e) {

      print(e); // Log para capturar o erro

    }

  }

}

* A classe FilmeController é a classe que faz a ponte entre a Service, que requisita a API, e a View;
* Por fim vamos para a pasta View criar as seguintes classes com seus códigos respectivamente:
  + *FilmeScreen.dart*:

import 'dart:io';

import 'package:flutter/material.dart';

import 'package:projeto\_filmes\_api/Controller/filme\_controller.dart';

import 'package:projeto\_filmes\_api/View/filme\_details\_screen.dart';

import 'package:projeto\_filmes\_api/View/filme\_edit\_screen.dart';

class FilmeScreen extends StatefulWidget {

  const FilmeScreen({super.key});

  @override

  State<FilmeScreen> createState() => \_FilmeScreenState();

}

class \_FilmeScreenState extends State<FilmeScreen> {

  // Controlador responsável por gerenciar os filmes

  final FilmeController \_controller = FilmeController();

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        // Define o título e os ícones da barra superior

        title: Text("Meus Filmes"),

        leading: Icon(Icons.movie), // Ícone de filme

        backgroundColor: Color.fromARGB(255, 110, 15, 15), // Cor de fundo do AppBar

        foregroundColor: Color.fromARGB(255, 226, 222, 222), // Cor do texto e ícones no AppBar

      ),

      body: Padding(

        padding: EdgeInsets.all(12.0), // Espaçamento ao redor da tela

        child: Column(

          children: [

            // Botão para adicionar um novo filme

            ListTile(

              title: Text("Adicionar um novo filme"),

              leading: Icon(Icons.add), // Ícone de adicionar

              onTap: () {

                // Navega para a tela de cadastro de filmes

                Navigator.pushNamed(context, '/cadastro');

              },

            ),

            Expanded(

              // Usado para expandir a lista de filmes para preencher o espaço disponível

              child: FutureBuilder(

                future: \_controller.getFromFilmeFromService(),

                builder: (context, snapshot) {

                  // Se a lista de filmes estiver vazia, exibe um indicador de progresso

                  if (\_controller.listFilmes.isEmpty) {

                    return Center(

                      child: CircularProgressIndicator(),

                    );

                  } else {

                    // Se houver filmes, exibe uma lista deles

                    return Center(

                      child: ListView.builder(

                        itemCount: \_controller.listFilmes.length, // Quantidade de filmes na lista

                        itemBuilder: (context, index) {

                          // Exibe informações básicas de cada filme

                          return ListTile(

                            title: Text(

                              \_controller.listFilmes[index].nome +

                                  " - (${\_controller.listFilmes[index].ano})",

                            ),

                            subtitle: Text(

                              "${\_controller.listFilmes[index].categoria} | ${\_controller.listFilmes[index].duracao} min",

                            ),

                            // Ícone de lixeira para excluir o filme

                            trailing: IconButton(

                              icon: Icon(Icons.delete),

                              onPressed: () async {

                                // Remove o filme do servidor e da lista

                                await \_controller.deleteFilme(

                                    \_controller.listFilmes[index].id!);

                                \_controller.listFilmes.removeAt(index);

                                setState(() {}); // Atualiza a interface após a exclusão

                              },

                            ),

                            onTap: () {

                              // Ao clicar no filme, navega para a tela de detalhes

                              Navigator.push(

                                context,

                                MaterialPageRoute(

                                  builder: (context) => FilmeDetailsScreen(

                                    filme: \_controller.listFilmes[index],

                                  ),

                                ),

                              );

                            },

                          );

                        },

                      ),

                    );

                  }

                },

              ),

            ),

          ],

        ),

      ),

    );

  }

}

* + *FilmeDetailsScreen.dart*:

import 'package:flutter/material.dart'; // Importa o pacote de widgets do Flutter

import 'package:projeto\_filmes\_api/Model/Filme.dart'; // Importa o modelo de Filme

// Tela de detalhes do filme

class FilmeDetailsScreen extends StatelessWidget {

  Filme filme; // Objeto filme que contém os detalhes a serem exibidos

  FilmeDetailsScreen({required this.filme}); // Recebe o objeto filme através do construtor

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Detalhes do Filme - ${filme.nome}"), // Exibe o nome do filme no título da AppBar

      ),

      body: Center(

        child: Column(

          children: [

            // Exibe os detalhes do filme como nome, ano, categoria, duração e classificação

            Text("Nome do filme: ${filme.nome}"),

            Text("Ano do filme: ${filme.ano}"),

            Text("Categoria do filme: ${filme.categoria}"),

            Text("Duração do filme: ${filme.duracao}min"),

            Text("Nota do filme: ${filme.classificacao} /10 ★"),

          ],

        ),

      ),

    );

  }

}

* + *CadastroScreen.dart*:

import 'dart:io';

import 'package:flutter/material.dart';

import 'package:flutter/services.dart';

import 'package:projeto\_filmes\_api/Controller/FilmeController.dart';

import 'package:projeto\_filmes\_api/Model/Filme.dart';

class CadastroScreen extends StatefulWidget {

  const CadastroScreen({super.key});

  @override

  State<CadastroScreen> createState() => \_CadastroScreenState();

}

class \_CadastroScreenState extends State<CadastroScreen> {

  final \_formKey = GlobalKey<FormState>(); // Chave para controlar o estado do formulário

  final FilmeController \_controller = FilmeController(); // Instância do controller de filmes

  // Controllers para capturar e controlar os inputs do formulário

  final TextEditingController \_nomeController = TextEditingController();

  final TextEditingController \_categoriaController = TextEditingController();

  final TextEditingController \_duracaoController = TextEditingController();

  final TextEditingController \_anoController = TextEditingController();

  final TextEditingController \_classificacaoController = TextEditingController();

  @override

  void initState() {

    // Chama o método que carrega filmes a partir do serviço ao inicializar a tela

    \_controller.getFromFilmeFromService();

    super.initState();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text('Cadastro'), // Título da página de cadastro

      ),

      body: Center(

        child: SingleChildScrollView( // Permite rolagem caso o conteúdo da tela seja maior que a tela

          child: Form(

            key: \_formKey, // Atribui a chave do formulário

            child: Padding(

              padding: EdgeInsets.all(15), // Espaçamento ao redor do formulário

              child: Column(

                mainAxisAlignment: MainAxisAlignment.center,

                children: <Widget>[

                  // Campo para o nome do filme

                  TextFormField(

                    decoration: InputDecoration(

                      labelText: 'Nome do filme',

                    ),

                    validator: (value) {

                      // Valida se o campo está vazio

                      if (value!.trim().isEmpty) {

                        return 'Por favor, insira o nome do filme';

                      } else {

                        return null;

                      }

                    },

                    controller: \_nomeController, // Controlador do input de nome

                  ),

                  SizedBox(height: 20), // Espaçamento entre os campos

                  // Campo para a categoria do filme

                  TextFormField(

                    decoration: InputDecoration(

                      labelText: 'Categoria do filme',

                    ),

                    validator: (value) {

                      if (value!.trim().isEmpty) {

                        return 'Por favor, insira a categoria do filme';

                      } else {

                        return null;

                      }

                    },

                    controller: \_categoriaController,

                  ),

                  SizedBox(height: 20),

                  // Campo para a duração do filme

                  TextFormField(

                    decoration: InputDecoration(

                      labelText: 'Duração',

                    ),

                    keyboardType: TextInputType.number, // Define o teclado numérico

                    inputFormatters: <TextInputFormatter>[

                      FilteringTextInputFormatter.digitsOnly // Aceita apenas dígitos

                    ],

                    validator: (value) {

                      if (value!.trim().isEmpty) {

                        return 'Por favor, insira a duração do filme em minutos';

                      } else {

                        return null;

                      }

                    },

                    controller: \_duracaoController,

                  ),

                  SizedBox(height: 20),

                  // Campo para o ano de lançamento

                  TextFormField(

                    decoration: InputDecoration(

                      labelText: 'Ano',

                    ),

                    keyboardType: TextInputType.number,

                    inputFormatters: <TextInputFormatter>[

                      FilteringTextInputFormatter.digitsOnly

                    ],

                    validator: (value) {

                      if (value!.trim().isEmpty) {

                        return 'Por favor, insira o ano de lançamento do filme';

                      } else {

                        return null;

                      }

                    },

                    controller: \_anoController,

                  ),

                  SizedBox(height: 20),

                  // Campo para a nota de classificação do filme

                  TextFormField(

                    decoration: InputDecoration(

                      labelText: 'Nota',

                    ),

                    keyboardType: TextInputType.number,

                    inputFormatters: <TextInputFormatter>[

                      FilteringTextInputFormatter.digitsOnly

                    ],

                    validator: (value) {

                      if (value!.trim().isEmpty) {

                        return 'Por favor, insira a sua classificação de 0 a 10 do filme';

                      } else {

                        return null;

                      }

                    },

                    controller: \_classificacaoController,

                  ),

                  SizedBox(height: 20),

                  // Botão para cadastrar o filme

                  ElevatedButton(

                    onPressed: () {

                      if (\_formKey.currentState!.validate()) {

                        \_cadastrarFilme(); // Chama a função de cadastro se o formulário for válido

                      }

                    },

                    child: Text("Cadastrar"),

                  ),

                ],

              ),

            ),

          ),

        ),

      ),

    );

  }

  // Função que cria um objeto Filme a partir dos dados inseridos no formulário

  Filme getFilmes() {

    return Filme(

      nome: \_nomeController.text,

      categoria: \_categoriaController.text,

      duracao: int.parse(\_duracaoController.text),

      ano: int.parse(\_anoController.text),

      classificacao: int.parse(\_classificacaoController.text),

    );

  }

  // Função para limpar os campos do formulário após o cadastro

  void limpar() {

    \_nomeController.clear();

    \_categoriaController.clear();

    \_duracaoController.clear();

    \_anoController.clear();

    \_classificacaoController.clear();

    setState(() {}); // Atualiza o estado para refletir as mudanças

  }

  // Função que valida e cadastra o filme, verificando se ele já está cadastrado

  void \_cadastrarFilme() {

    bool verificaFilme = false;

    // Verifica se o filme já existe na lista de filmes cadastrados

    for (int i = 0; i < \_controller.listFilmes.length; i++) {

      if (\_nomeController.text.trim().toLowerCase() ==

          \_controller.listFilmes[i].nome.trim().toLowerCase()) {

        verificaFilme = true;

        break;

      }

    }

    // Se o filme não existir, cadastra o novo filme

    if (!verificaFilme) {

      \_controller.addFilme(getFilmes()); // Adiciona o filme ao controller

      ScaffoldMessenger.of(context).showSnackBar(

        const SnackBar(

          content: Text('Filme cadastrado com sucesso!'),

          duration: Duration(seconds: 2),

        ),

      );

      limpar(); // Limpa o formulário após o cadastro bem-sucedido

      setState(() {

        Navigator.pushNamed(context, '/home'); // Navega para a tela inicial após o cadastro

      });

    } else {

      // Se o filme já estiver cadastrado, exibe uma mensagem de erro

      ScaffoldMessenger.of(context).showSnackBar(

        const SnackBar(

          content: Text('Filme já cadastrado!'),

          duration: Duration(seconds: 2),

        ),

      );

    }

  }

}

* Por fim edite o arquivo main.dart:

import 'package:flutter/material.dart';

import 'package:projeto\_filmes\_api/View/CadastroScreen.dart'; // Importa a tela de cadastro

import 'package:projeto\_filmes\_api/View/FilmeScreen.dart'; // Importa a tela principal de filmes

void main() {

  runApp(MyApp()); // Inicializa o aplicativo chamando a classe MyApp

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: "Projeto Filmes API", // Define o título do aplicativo

      home: FilmeScreen(), // Define a tela inicial como FilmeScreen

      debugShowCheckedModeBanner: false, // Remove o banner de debug

      routes: {

        '/home': (context) => FilmeScreen(), // Rota para a tela de filmes

        '/cadastro': (context) => CadastroScreen() // Rota para a tela de cadastro

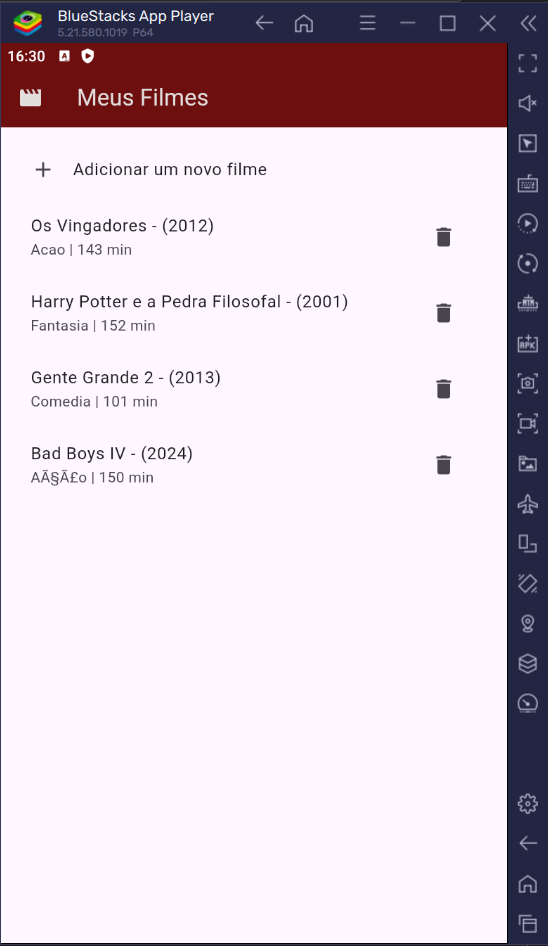
      },

    );

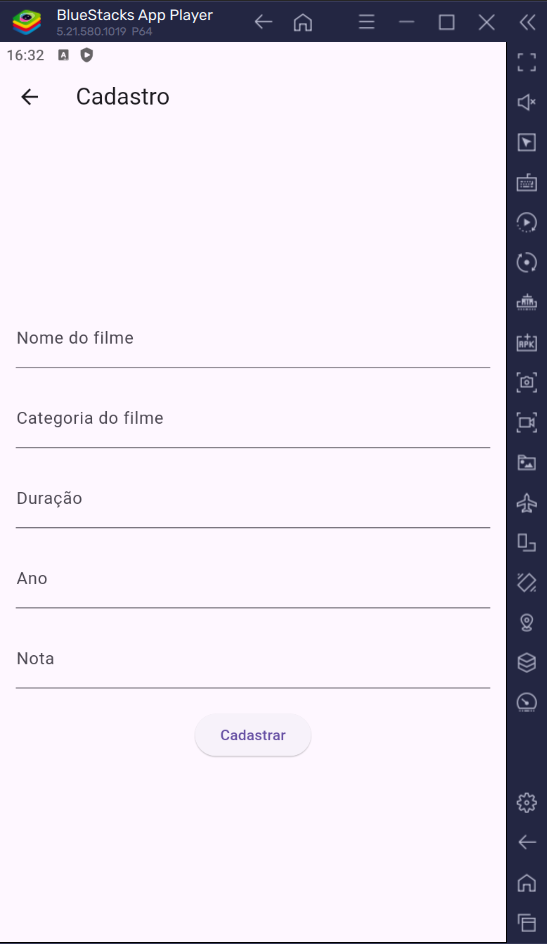
  }

}

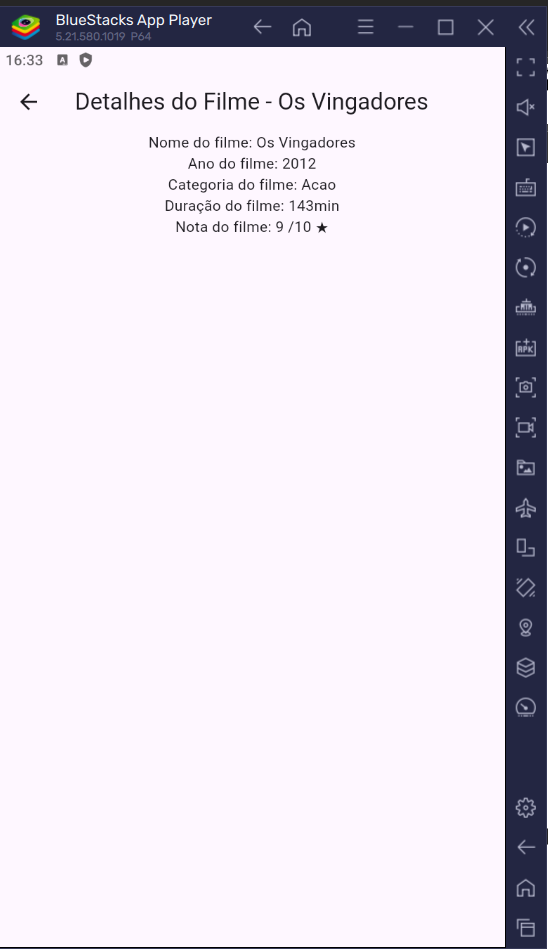
* Agora vamos rodar a aplicação!
* **Resultado Esperado:**
* *1 - Tela inicial*



* *2 – Cadastro*



* *3 – Detalhes*



E dessa maneira fizemos uma aplicação básica integrando API e Aplicação Flutter. Gostaria de agradece a você que esteve até, espero que tenha gostado do tutorial e te ajudado a entender mais sobre APIs e como elas podem ser versáteis.