

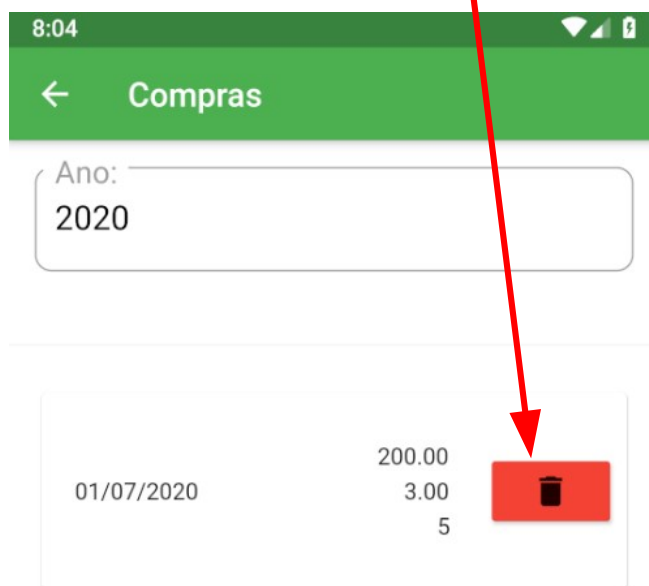
Shared Preferences, PopupMenuButton e Font Awesome



Usando ícones

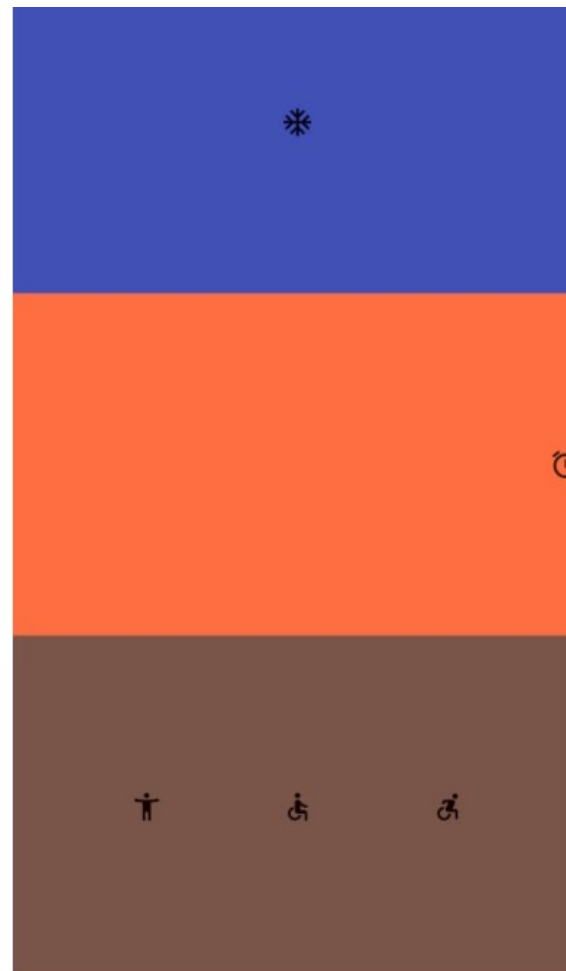
- Quando fazemos menus, botões, etc é comum utilizarmos ícones para deixar a aplicação com um visual melhor e mais intuitivo.
- A biblioteca padrão para uso de ícones no Flutter é a **icons.dart**.
- Essa biblioteca é acessada pela importação de **material.dart** (`import 'package:flutter/material.dart';`). Isso porque **material.dart** faz o apontamento para **icons.dart**.
- Ao lado vemos o uso do ícone **delete** da biblioteca **icons.dart** no app **Fils Plan** (tela de Compra de Fundos Imobiliários).

```
77 Expanded(  
78   flex: 3,  
79   child: BotaoIcone(  
80     // texto: "Excluir",  
81     ao_clicar: () async {  
82       controle.removerCompra(index);  
83     },  
84     cor: Colors.red,  
85     icone: Icons.delete,  
86   ), // BotaoIcone  
87 ), // Expanded
```



Usando ícones

```
50 _body() {  
51   return Container(  
52     child: Column(  
53       children: <Widget>[  
54         Expanded(  
55           child: Container(  
56             color: Colors.indigo,  
57             alignment: Alignment.center,  
58             child: Icon(Icons.ac_unit)), // Container, Expanded  
59         ),  
60         Expanded(  
61           child: Container(  
62             color: Colors.deepOrangeAccent,  
63             alignment: Alignment.centerRight,  
64             child: Icon(Icons.access_alarm)), // Container, Expanded  
65         ),  
66         Expanded(  
67           child: Container(  
68             color: Colors.brown,  
69             alignment: Alignment.centerLeft,  
70             child: Row(  
71               mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
72               children: <Widget>[  
73                 Icon(Icons.accessibility_new),  
74                 Icon(Icons.accessible),  
75                 Icon(Icons.accessible_forward),  
76               ], // <Widget>[]  
77             ), // Row  
78           ), // Container  
79         ), // Expanded  
80       ], // <Widget>[]  
81     ), // Column  
82   ); // Container  
83 }
```



- Já havíamos utilizado a biblioteca de ícones na aula de Componentes Visuais Básicos – Parte 2.
- O widget **Icon** foi utilizado internamente em **Botaolcone** do slide anterior.

Font Awesome

- Mas e se precisarmos de um ícone que não existe em **icons.dart**. Ou se o que existe for “feio” ou inadequado.
- Para tentar resolver esse problema, fornecendo mais opções de ícones, temos **Font Awesome**.
- Para instalar o plugin precisaremos ir ao site **pub.dev**.
- Assim como outros plugins, a instalação é bem simples e o processo se encontra descrito em:
https://pub.dev/packages/font_awesome_flutter#-installing-tab-
- É importante nunca esquecer de fazer o **flutter pub get** para que o pacote seja colocado no projeto.
- No próximo slide veremos o uso de um ícone do plugin **Font Awesome**.

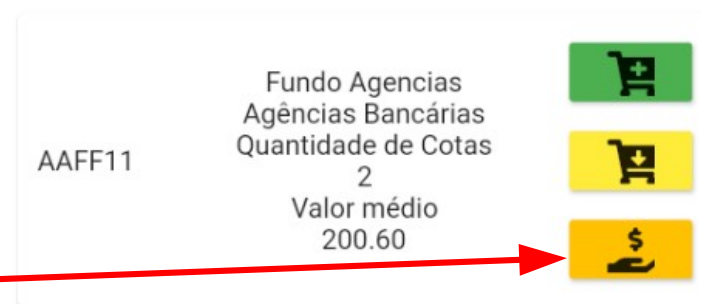


Font Awesome


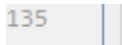
- Todos os ícones dessa tela vieram de Font Awesome.
- Nesse slide vemos o código do botão que leva para a Tela de Dividendos no FIIs Plan.

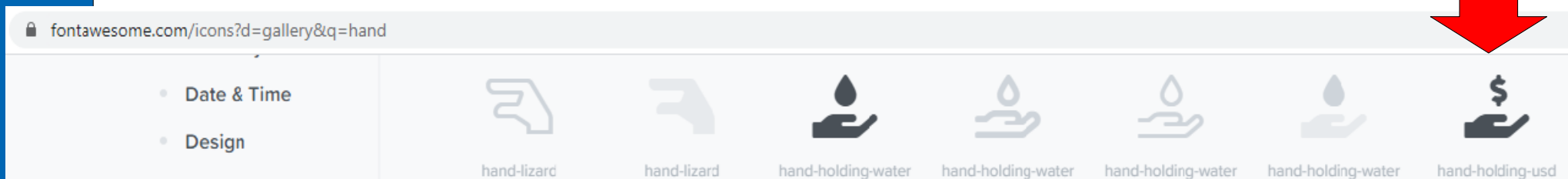


```
— Expanded(  
  flex: 1,  
  child: BotaoIcone(  
    ao_clicar: () async {  
      await controle.irParaTelaDividendos(context, patrimonio);  
    },  
    cor: Colors.amber,  
    icone: FontAwesomeIcons.handHoldingUsd,  
  ), // BotaoIcone  
) // Expanded
```



Site Font Awesome

- Uma dificuldade de se utilizar o Font Awesome é conseguir saber como é o desenho de cada ícone. Com a biblioteca padrão (**icons.dart**) o Android Studio consegue mostrar o ícone (o que facilita muito a visualização).  `icone: Icons.delete,`
- Mas quando usamos o Font Awesome não temos essa visualização.  `icone: FontAwesomeIcons.handHoldingUsd,`
- Para conseguir ver os ícones de Font Awesome podemos ir no site <https://fontawesome.com/icons?d=gallery&q=hand>.



Shared Preferences

- **Shared Preferences** é uma forma simples de persistir dados em um app.
- Trabalha armazenando informações utilizando **chave x valor**.
- Um uso muito comum de **Shared Preferences** é obter o usuário logado em qualquer tempo. Mas pode ser usado também para armazenar, por exemplo, a última Tab que um usuário navegou (para voltar a ela de forma automática quando o usuário usar o app novamente).
- No app FIIs Plan utilizamos Shared Preferences para obter o usuário logado em qualquer tempo. Para tanto foram implementados 3 métodos, um para salvar o usuário em Shared Preferences, outro para obtê-lo quando necessário e o terceiro para apagar o usuário quando não for mais utilizado. Nos próximos slides veremos esse código da classe Usuario.



Salvar um Usuário em Shared Preferences

```
// Salvando o usuário em Shared Preferences
void salvar(){
    // Transformando o usuário em map
    Map usuario_map = this.toMap();

    // Transformando a map em String
    String usuario_string = json.encode(usuario_map);

    // Armazenando o usuário em Shared Preferences
    Prefs.setString("user.prefs", usuario_string);
}
```

```
Map<String, dynamic> toMap(){
    final Map<String, dynamic> data = Map<String, dynamic>();
    data['id'] = this.id;
    data['nome'] = this.nome;
    data['tipo'] = this.tipo;
    data['login'] = this.login;
    data['senha'] = this.senha;
    data['endereco'] = this.endereco;
    data['urlFoto'] = this.urlFoto;
    return data;
}
```

- Aqui o usuário é convertido primeiramente num Map.
- Na sequência transformamos o Map numa String.
- Por fim armazenamos com a chave “**user.prefs**” a String que equivale ao usuário.
- **Prefs** é uma classe utilitária do FFI's Plan. Essa classe encapsula o **Shared Preferences** deixando-o mais simples.
- A importação `import 'dart:convert';` é feita na classe Usuario para uso do método **encode** (para converter de Map para String).



Obter um Usuário usando Shared Preferences

```
static Future<Usuario> obter() async{  
  String usuario_string = await Prefs.getString("user.prefs");  
  if (usuario_string.isEmpty){  
    return null;  
  }  
  
  Map usuario_map = json.decode(usuario_string);  
  
  Usuario usuario = Usuario.fromMap(usuario_map);  
  return usuario;  
}
```

```
Usuario.fromMap(Map<String, dynamic> map) : super.fromMap(map){  
  nome = map["nome"];  
  tipo = map["tipo"];  
  login = map["login"];  
  senha = map["senha"];  
  endereco = map["endereco"];  
  urlFoto = map["urlFoto"];  
}
```

- Aqui, usando a chave “**user.prefs**” obtemos a String que possui os dados necessários para a criação de um objeto Usuario.
- Se a String está vazia, o usuário não existe em **Shared Preferences** (nesse caso não haveria usuário logado).
- Na sequência convertemos a String num Map usando o método **decode**.
- Por fim criamos um Usuario a partir do Map usando o construtor nomeado **fromMap**.



Flutter 2.0

```
static Future<Usuario> obter() async{
  String usuario_string = await Prefs.getString("user.prefs");
  if (usuario_string.isEmpty){
    return null;
  }

  Map usuario_map = json.decode(usuario_string);

  Usuario usuario = Usuario.fromMap(usuario_map);
  return usuario;
}
```

```
static Future<Usuario> obterNaoNulo() async{
  String usuario_string = await Prefs.getString("user.prefs");

  Map<String, dynamic> usuario_map = json.decode(usuario_string);

  Usuario usuario = Usuario.fromMap(usuario_map);
  return usuario;
}
```

```
static Future<Usuario?> obter() async{
  String usuario_string = await Prefs.getString("user.prefs");
  if (usuario_string.isEmpty){
    return null;
  }

  Map<String, dynamic> usuario_map = json.decode(usuario_string);

  Usuario usuario = Usuario.fromMap(usuario_map);
  return usuario;
}
```

Precisei de uma versão que retornasse Future<Usuario> para uso em um FutureBuilder (ele não aceitava Future<Usuario?>)



Excluir um Usuário de Shared Preferences

- A exclusão de um Usuário é muito simples. Seu código pode ser visto abaixo:

```
static void limpar(){  
    // Limpando o usuário em Shared Preferences  
    Prefs.setString("user.prefs", "");  
}
```



Prefs

- Aqui temos o código da classe **Prefs**.
- Basicamente, para cada tipo específico, é obtida a instância de **SharedPreferences** e utilizado **gets** usando a chave informada ou **sets** passando a chave e o valor.
- No nosso caso, apenas utilizamos os métodos **getString** e **setString**.
- Essa classe utilitária foi obtida do curso de Flutter Essencial do professor Ricardo Lecheta no Udemy:
<https://www.udemy.com/course/flutter-essencial/>.
Recomendo esse curso para quem desejar complementar seus estudos.

```
class Prefs {  
  
    static Future<bool> getBool(String key) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        return prefs.getBool(key) ?? false;  
    }  
  
    static void setBool(String key, bool b) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        prefs.setBool(key, b);  
    }  
  
    static Future<int> getInt(String key) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        return prefs.getInt(key) ?? 0;  
    }  
  
    static void setInt(String key, int i) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        prefs.setInt(key, i);  
    }  
  
    static Future<String> getString(String key) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        return prefs.getString(key) ?? "";  
    }  
  
    static void setString(String key, String s) async {  
        var prefs = await SharedPreferences.getInstance();  
  
        prefs.setString(key, s);  
    }  
}
```

Usando os métodos de Usuario

- Aqui fizemos uso do método **obter** para tentar obter o Usuario logado.
- Caso haja Usuario logado, ele será obtido a partir do **futureC**, cujo valor foi retornado na **posição 2** da lista **values**.
- Se houver Usuario logado é necessário saber seu tipo (regra do FII's Plan). Se for **padrao** vai para a **TelaPrincipal** para trabalhar com Fundos Imobiliários. Se for **administrador** vai para a **TelaAdministracaoUsuario** para trabalhar com Usuarios.
- Se não houver Usuario logado será chamada a **TelaLogin**.

```
class ControleTelaAbertura{  
  void inicializarAplicacao(BuildContext context) {  
  
    // Inicializando o banco  
    Future futureA = DatabaseHelper.getInstance().db;  
  
    // Dando um tempo para exibição da tela de abertura  
    Future futureB = Future.delayed(Duration(seconds: 3));  
  
    // Obtendo o usuário logado (se houver)  
    Future<Usuario> futureC = Usuario.obter();  
  
    // Agurandando as 3 operações terminarem  
    // Quando terminarem a aplicação ou vai para a tela de login  
    // ou para a tela principal  
    Future.wait([futureA, futureB, futureC]).then((List values) {  
      Usuario usuario = values[2];  
  
      if(usuario != null){  
        if (usuario.tipo == TipoUsuario.padrao) {  
          push(context, TelaPrincipal(usuario), replace: true);  
        } else {  
          push(context, TelaAdministracaoUsuario(), replace: true);  
        }  
      } else {  
        push(context, TelaLogin(), replace: true);  
      }  
    });  
  }  
}
```

Esse código é chamado pela primeira Tela do App, a TelaAbertura



Usando os métodos de Usuario

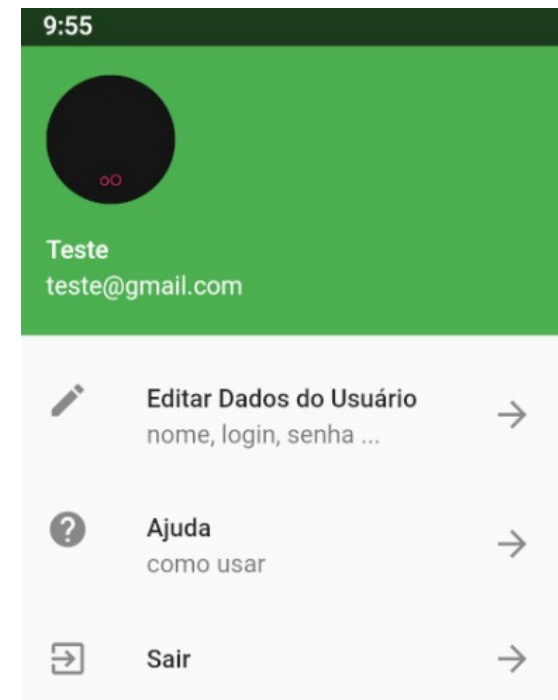
```
class ControleTelaLogin {  
  // Controles de edição do login e senha  
  final controlador_login = TextEditingController();  
  final controlador_senha = TextEditingController();  
  
  // Controlador de formulário (para fazer validações)  
  final formkey = GlobalKey<FormState>();  
  
  // Controladores de foco  
  final focus_senha = FocusNode();  
  final focus_botao = FocusNode();  
  
  void logar(BuildContext context) async{  
    if (formkey.currentState.validate()){  
      String login = controlador_login.text.trim();  
      String senha = controlador_senha.text.trim();  
      Usuario usuario = await FabricaControladora.obterUsuarioControl().obterUsuario(login, senha);  
      if(usuario != null){  
        // Guardando em Shared Preferences  
        usuario.salvar();  
        if (usuario.tipo == TipoUsuario.padrao){  
          push(context, TelaPrincipal(usuario), replace: true);  
        } else {  
          push(context, TelaAdministracaoUsuario(), replace: true);  
        }  
      } else {  
        MensagemAlerta("Login ou senha inválidos!!");  
      }  
    }  
  }  
}
```

- Aqui temos a classe de Controle da Tela de Login.
- O método **logar** faz a validação dos dados preenchidos na **TelaLogin** e tenta obter o **Usuario** no banco de dados do aplicativo (conteúdo de outra aula).
- Posteriormente esse **Usuario** é persistido em **Shared Preferences** através do método **salvar**.
- Da próxima vez que o usuário utilizar o app, ele estará logado e não precisará entrar na **TelaLogin** para informar login e senha.



Usando os métodos de Usuário

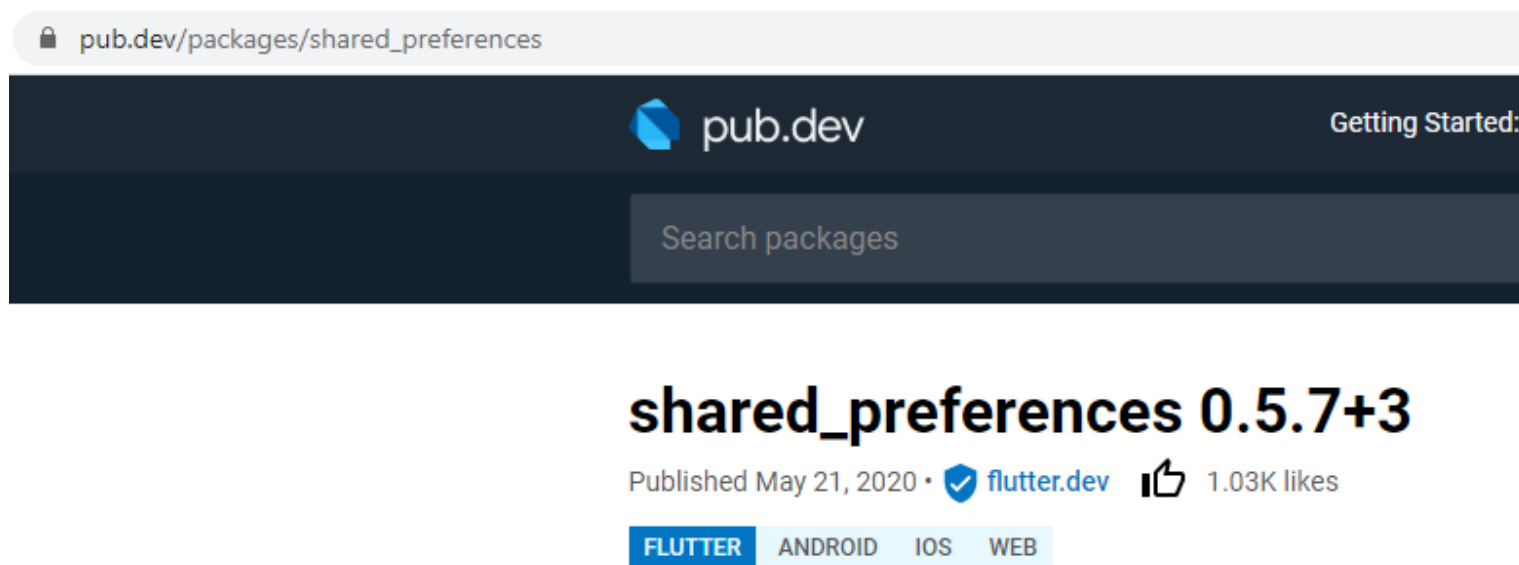
- Aqui temos o menu lateral (a ser estudado em outra aula) e a opção **Sair**.
- Essa opção fecha o menu lateral (o menu lateral é considerado uma tela, por isso o **pop**) e volta para a **TelaLogin** (usando o método **push** que irá sobrescrever a **TelaPrincipal**).
- Finalmente temos a chamada do método **limpar** para retirar o Usuário de **Shared Preferences** (isso irá obrigar o usuário a se logar novamente para utilizar o aplicativo Fils Plan).



```
ListTile(  
  leading: Icon(Icons.exit_to_app),  
  title: Text("Sair"),  
  trailing: Icon(Icons.arrow_forward),  
  onTap: () {  
    // Fechando o menu lateral  
    pop(context);  
  
    // Sobrescrevendo a tela de Login  
    push(context, TelaLogin(), replace: true);  
  
    // Retirando o usuário das Shared Preferences  
    Usuario.limpar();  
  },  
) // ListTile
```

Instalação do plugin

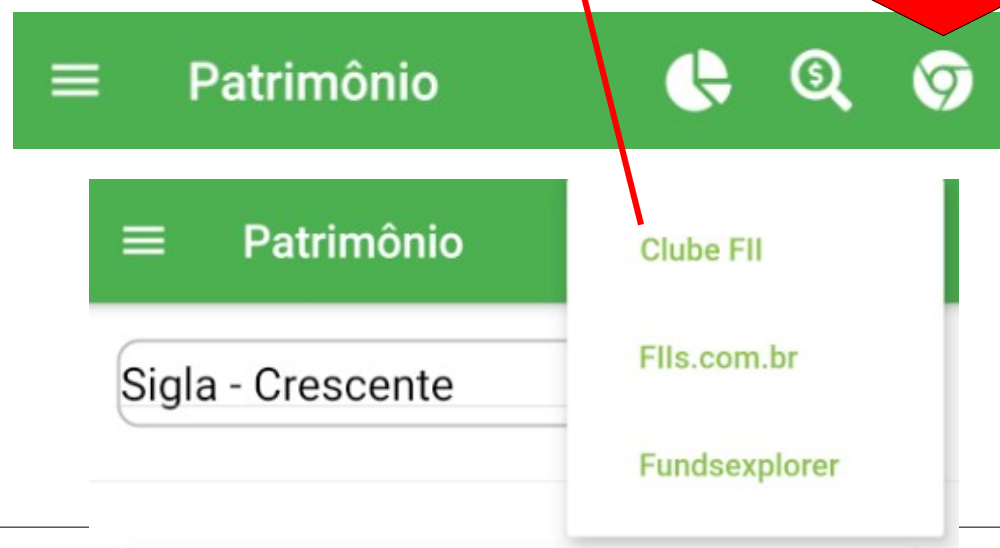
- O plugin, como de costume, se encontra em **pub.dev**.
- O processo de instalação se encontra em:
https://pub.dev/packages/shared_preferences#-installing-tab-
(basicamente é o mesmo procedimento que fizemos para instalar outros plugins).



PopupMenuButton

- O **PopupMenuButton** mostra um menu de opções a serem selecionadas.
- Normalmente ele vem com o ícone de “tres pontos verticais” mas é possível customizar (como foi feito no FII's Plan colocando o ícone do Chrome).
- Aqui colocamos o **PopupMenuButton** no **actions** da **AppBar** da **TelaPrincipal** do FII's Plan, mas esse widget não precisa, obrigatoriamente, ser usado no contexto de um **AppBar**.

```
PopupMenuButton<String>(  
  onPressed: (String valor) {  
    push(context, TelaWebViewFundos(valor));  
  },  
  icon: FaIcon(FontAwesomeIcons.chrome),  
  itemBuilder: (BuildContext context) {  
    return [  
      PopupMenuItem(  
        value: "https://www.clubefii.com.br/fundo_imobiliario_lista",  
        child: FlatButton(  
          child: Text(  
            "Clube FII",  
            style: TextStyle(  
              color: Colors.lightGreen,  
            ), // TextStyle  
          ), // Text  
        ), // FlatButton  
      ), // PopupMenuItem  
    ]  
  )  
)
```



```
PopupMenuButton<String>(  
  |  onSelected: (String valor) {  
  |  |  push(context, TelaWebViewFundos(valor));  
  |  },  
  |  icon: FaIcon(FontAwesomeIcons.chrome),
```

Clube FII

FIIIs.com.br

Fundsexplorer

PopupMenuButton

```
PopupMenuItem(  
  |  value: "https://fiis.com.br/lista-de-fundos-imobiliarios/",  
  |  child: FlatButton(  
  |  |  child: Text(  
  |  |  |  "FIIIs.com.br",  
  |  |  |  style: TextStyle(  
  |  |  |  |  color: Colors.lightGreen,  
  |  |  |  ), // TextStyle  
  |  |  ), // Text  
  |  ), // FlatButton  
  |  ), // PopupMenuItem  
  |  PopupMenuItem(  
  |  |  value: "https://www.fundsexplorer.com.br/funds",  
  |  |  child: FlatButton(  
  |  |  |  child: Text(  
  |  |  |  |  "Fundsexplorer",  
  |  |  |  |  style: TextStyle(  
  |  |  |  |  |  color: Colors.lightGreen,  
  |  |  |  |  ), // TextStyle  
  |  |  |  ), // Text  
  |  |  ), // FlatButton  
  |  |  ), // PopupMenuItem  
  |  ],  
  |  },  
  ) // PopupMenuButton
```

- Um **PopupMenuItem** é um item do menu de **PopupMenuButton**.
- Usamos um **FlatButton** e um **Text** para representar visualmente cada item do menu.
- O valor **value** é passado para a função anônima definida em **onSelected**. Então, no parâmetro **valor**, recebemos o endereço do site de Fundos Imobiliários para o qual deseja-se ir.
- O **push** então chama **TelaWebViewFundos** que basicamente é uma tela que utiliza uma **WebView** (conteúdo de outra aula) para apresentar uma página web selecionada.
- Apenas para ilustrar: Se a linha **icon: Falcon(FontAwesomeIcons.chrome)** for comentada teremos o resultado abaixo (ícone dos “três pontos”):



Flutter 2.0

```
PopupMenuButton<String>(  
  onSelect: (String valor) {  
    push(context, TelaWebViewFundos(valor));  
  },  
  icon: FaIcon(FontAwesomeIcons.chrome),  
  itemBuilder: (BuildContext context) {  
    return [  
      PopupMenuItem(  
        value: "https://www.clubefii.com.br/fundo_imobiliario_lista",  
        child: FlatButton(  
          child: Text(  
            "Clube FII",  
            style: TextStyle(  
              color: Colors.lightGreen,  
            ), // TextStyle  
          ), // Text  
        ), // FlatButton  
      ), // PopupMenuItem  
    ]  
  }  
)
```

Saiu daqui!!!

```
- PopupMenuItem(  
  child: TextButton(  
    onPressed: () { push(context, TelaWebViewFundos("https://fiis.com.br/lista-de-fundos-imobiliarios/")); },  
    child: Text(  
      "FIIs.com.br",  
      style: TextStyle(  
        color: Colors.lightGreen,  
      ), // TextStyle  
    ), // Text  
  ), // TextButton  
) , // PopupMenuItem  
- PopupMenuItem(  
  child: TextButton(  
    onPressed: () { push(context, TelaWebViewFundos("https://www.fundsexplorer.com.br/funds")); },  
    child: Text(  
      "Fundsexplorer",  
      style: TextStyle(  
        color: Colors.lightGreen,  
      ), // TextStyle  
    ), // Text  
  ), // TextButton  
) , // PopupMenuItem
```

Foi para cada opção

```
PopupMenuButton<String>(  
  icon: FaIcon(FontAwesomeIcons.chrome),  
  itemBuilder: (BuildContext context) {  
    return [  
      PopupMenuItem(  
        child: TextButton(  
          onPressed: () { push(context, TelaWebViewFundos("https://www.clubefii.com.br/fundo_imobiliario_lista")); },  
          child: Text(  
            "Clube FII",  
            style: TextStyle(  
              color: Colors.lightGreen,  
            ), // TextStyle  
          ), // Text  
        ), // TextButton  
      ), // PopupMenuItem  
    ]  
  }  
)
```



Dúvidas?

