

Aplicativos de Foto e Galeria



Plugin image_picker

- Para chamarmos os aplicativos de foto e galeria precisaremos do plugin **image_picker**.
- O plugin image_picker se encontra no site **pub.dev** e as instruções de instalação se encontram no link:
https://pub.dev/packages/image_picker#-installing-tab-



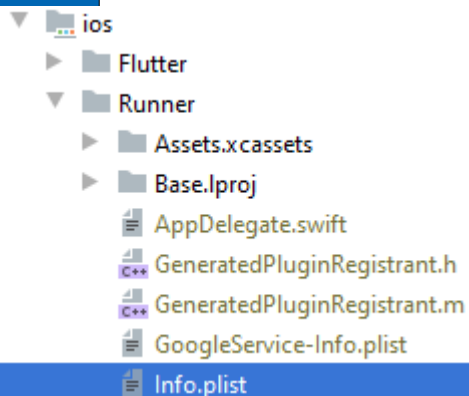
Configurações Complementares iOS

- Essas informações se encontram no link:

https://pub.dev/packages/image_picker#-readme-tab-

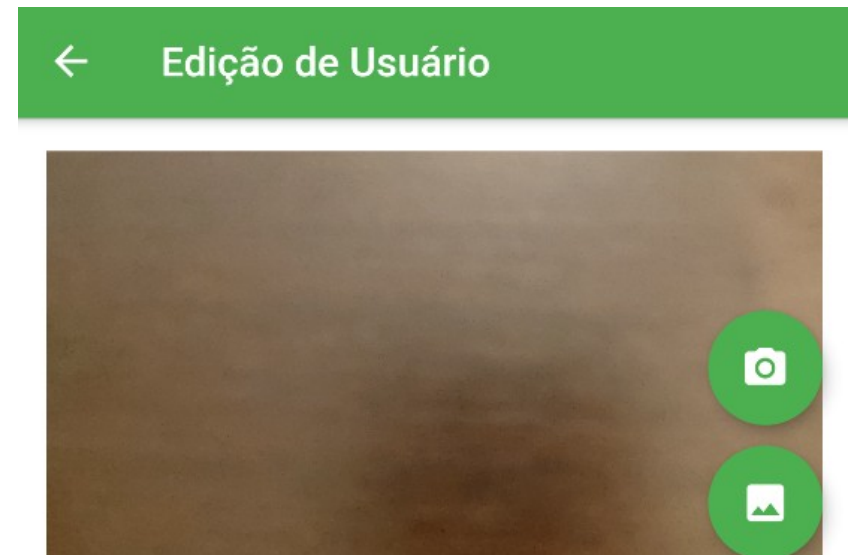
- Basicamente solicitamos permissões para acesso a galeria, câmera e microfone (poderia ser vídeo, por exemplo).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>io.flutter.embedded_views_preview</key>
    <true/>
    <key>NSPhotoLibraryUsageDescription</key>
    <string>O aplicativo necessita acesso a galeria</string>
    <key>NSCameraUsageDescription</key>
    <string>O aplicativo necessita acesso a camera </string>
    <key>NSMicrophoneUsageDescription</key>
    <string>O aplicativo necessita acesso ao microfone</string>
    <key>CFBundleDevelopmentRegion</key>
    <string>$(DEVELOPMENT_LANGUAGE)</string>
</dict>
</plist>
```



Contexto do uso

- Utilizamos o plugin **image_picker** para capturar imagens da câmera e da galeria na tela de cadastro de usuário do app Flls Plan.
- Ao clicar no botão de foto será acionado o aplicativo de câmera. Se for tirada uma foto e confirmada, a foto irá ocupar o espaço previsto para ela.
- Da mesma forma, ao clicar no botão de galeria o aplicativo de galeria será chamado e a imagem selecionada irá substituir a imagem padrão (da logo do app).
- No próximo slide veremos o código responsável por desenhar esse pedaço de tela ao lado.



TelaEdicaoUsuario

```
Container _imagem_botoes() {  
  return Container(  
    height: 200,  
    margin: EdgeInsets.only(bottom: 16),  
    child: Stack(  
      fit: StackFit.expand,  
      children: <Widget>[  
        _controle.imagem != null  
          ? Image.file(  
            _controle.imagem,  
            fit: BoxFit.cover,  
          ) // Image.file  
        : Image.asset(  
            "assets/icon/icone_aplicacao.png",  
            fit: BoxFit.contain,  
          ), // Image.asset  
      ],  
    ),  
  );  
}
```

- Foi feito um método especificamente para desenhar a imagem e os botões no espaço disponibilizado (height: 200).
- Optou-se pelo uso de uma **Stack** para podermos sobrepor os botões sobre a imagem.
- A Stack irá ocupar todo o espaço disponível (StackFit.expand).
- **_controle.imagem** é o arquivo que contém a imagem a ser exibida. Se for nulo significa que o usuário não possui imagem, então nesse contexto, é buscada uma imagem padrão na pasta **assets**. Se não for nulo é porque existe imagem selecionada para o usuário, então essa imagem é utilizada.



Botão Foto



- A chamada do código do **ImagePicker** é muito simples e o retorno é o arquivo de imagem. Nesse contexto, caso não seja nulo (tenha sido tirada uma foto e confirmada), essa imagem é atualizada na tela.
- **heroTag** é necessário pois coloquei mais de um **FloatingActionButtton** na mesma tela (câmera e galeria). Para maiores informações:

<https://api.flutter.dev/flutter/material/FloatingActionButton/heroTag.html>

```
FloatingActionButton(  
  heroTag: "BotaoCamera",  
  onPressed: () async {  
    var image =  
      await ImagePicker.pickImage(source: ImageSource.camera);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = image;  
      });  
    }  
  },  
)
```



Flutter 2.0

```
FloatingActionButton(  
  heroTag: "BotaoCamera",  
  onPressed: () async {  
    var image =  
      await ImagePicker.pickImage(source: ImageSource.camera);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = image;  
      });  
    }  
  },  
),
```

```
static File XFileToFile(XFile arquivoX){  
  return File(arquivoX.path);  
}
```

image passou a ser um XFile
ao invés de File.
Ou seja, mudou o retorno
do método pickImage

```
FloatingActionButton(  
  heroTag: "BotaoCamera",  
  onPressed: () async {  
    var image =  
      await ImagePicker().pickImage(source: ImageSource.camera);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = GerenciadoraArquivo.XFileToFile(image);  
      });  
    }  
  },  
),
```



Botão de Galeria



- O código para Galeria é muito similar ao código utilizado para foto. Mudando basicamente o parâmetro **source**.

```
FloatingActionButton(  
  heroTag: "BotaoGaleria",  
  onPressed: () async {  
    var image = await ImagePicker.pickImage(  
      source: ImageSource.gallery);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = image;  
      });  
    }  
  },  
)
```



Flutter 2.0

```
FloatingActionButton(  
  heroTag: "BotaoGaleria",  
  onPressed: () async {  
    var image = await ImagePicker.pickImage(  
      source: ImageSource.gallery);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = image;  
      });  
    }  
  },  
);
```

```
static File XFileToFile(XFile arquivoX){  
  return File(arquivoX.path);  
}
```

image passou a ser um XFile
ao invés de File.
Ou seja, mudou o retorno
do método pickImage

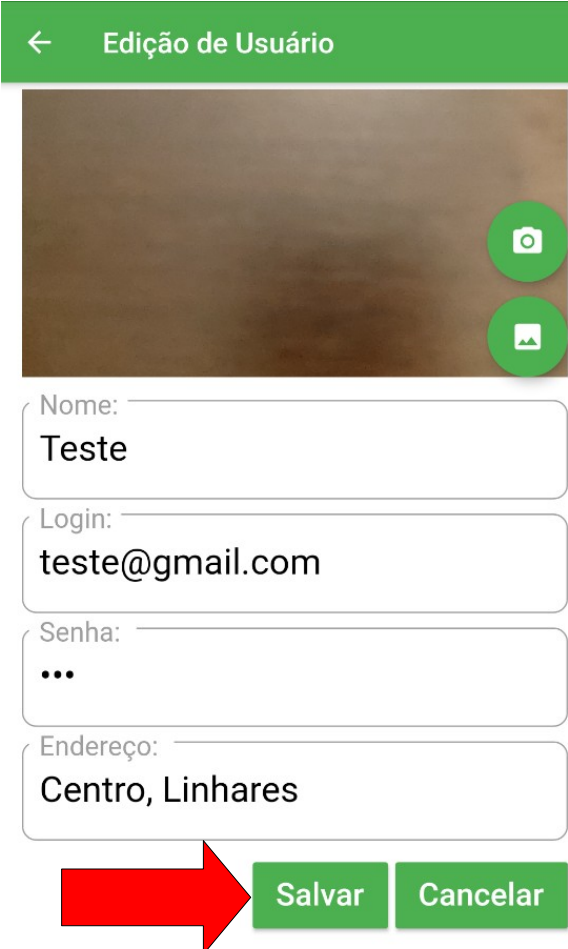
```
- FloatingActionButton(  
  heroTag: "BotaoGaleria",  
  onPressed: () async {  
    var image = await ImagePicker().pickImage(  
      source: ImageSource.gallery);  
    if (image != null) {  
      setState(() {  
        _controle.imagem = GerenciadoraArquivo.XFileToFile(image);  
      });  
    }  
  },  
);
```



Persistência do arquivo de imagem

- Utilizar os aplicativos de foto e galeria é muito simples, mas há uma questão que ainda não foi respondida: Como esses arquivos são persistidos? Da forma mostrada até agora, não são, ao fechar a tela esse arquivo simplesmente seria perdido.
- Nesse contexto, o botão **Salvar** terá de fazer o trabalho de persistir os dados de todos os campos e também da imagem. Vejamos no próximo slide como o Controlador faz isso.

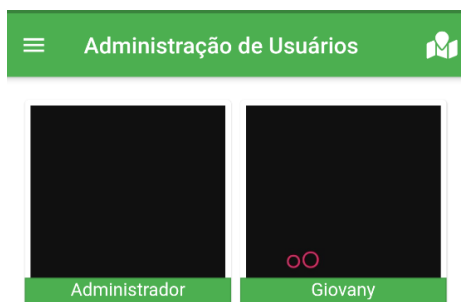
```
Botao(  
  texto: "Salvar",  
  cor: Colors.green,  
  ao_clicar: () {  
    _controle.salvar_usuario(context);  
  },  
  marcador_foco: _controle.focus_botao_salvar,  
), // Botao
```



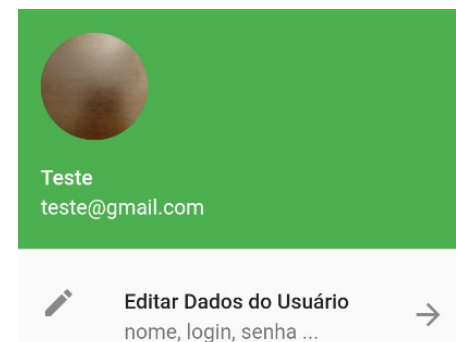
Persistindo a imagem

- O primeiro passo é a validação do formulário (os demais campos da tela estão todos dentro de um formulário).
- Se tudo estiver “ok” ocorre a chamada a **atualizar_usuario** e quando esse terminar a **TelaEdicaoUsuario** é desempilhada avisando ao **ControladorAdministracaoUsuario** (caso tenha vindo da **TelaAdministracaoUsuario**, se tivermos vindo de **MenuLateral**, simplesmente o retorno não será utilizado) que ocorreu um Salvamento (retorno “**Salvou**”). Esse por sua vez chama **buscarUsuarios** que irá atualizar a listagem na **TelaAdministracaoUsuario**.

```
void salvar_usuario(BuildContext context){  
  if (formkey.currentState.validate()) {  
    Future future = atualizar_usuario(context);  
    future.then((value){  
      Navigator.pop(context, "Salvou");  
    });  
  }  
}
```



```
ListTile(  
  leading: Icon(Icons.edit),  
  title: Text("Editar Dados do Usuário"),  
  subtitle: Text("nome, login, senha ..."),  
  trailing: Icon(Icons.arrow_forward),  
  onTap: () async {  
    // Fechando o menu Lateral  
    pop(context);  
    push(context, TelaEdicaoUsuario(usuario));  
  }  
) , // ListTile
```



```
void irParaTelaEdicaoUsuario(BuildContext context, Usuario usuario) async{  
  String s = await push(context, TelaEdicaoUsuario(usuario));  
  if (s == "Salvou"){  
    buscarUsuarios();  
  }  
}
```

Atualizando a TelaAdministracaoUsuario



Persistindo a imagem

```
Future<bool> atualizar_usuario(BuildContext context) async{
  if(usuario != null){
    usuario.nome = controlador_nome.text;
    usuario.login = controlador_login.text;
    usuario.senha = controlador_senha.text;
    usuario.endereco = controlador_endereco.text;
    usuario.tipo = tipo_usuario_selecionado;
    if(imagem != null){
      // Se já havia foto pode ser necessário apagá-la
      if (usuario.urlFoto != null){
        // Se houve troca de foto
        if (imagem.path != usuario.urlFoto){
          GerenciadoraArquivo.excluirArquivo(usuario.urlFoto);
          usuario.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem);
        }
        // Se não havia foto é necessário salvá-la
      } else {
        usuario.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem);
      }
    }
    // Se o usuário logado é o que está sendo salvo
    // atualizamos as Shared Preferences
    if(usuario_logado.id == usuario.id) {
      usuario.salvar();
    }
  }
  FabricaControladora.obterUsuarioControl().atualizarUsuario(usuario);
}
```

- **usuario != null** significa que o usuário não é novo, ou seja, é uma atualização. Na sequência vemos os atributos do usuário serem atualizados com os dados dos campos da tela.
- **imagem != null** significa que uma foto foi tirada ou uma imagem da galeria foi selecionada. Se o usuário já tinha uma foto será necessário apagá-la antes de salvar a imagem.
- Se não havia imagem no usuário basta salvar a nova imagem.
- **GerenciadoraArquivo** será vista posteriormente. Ela foi criada para persistir/gerenciar arquivos no dispositivo.
- Se o usuário é o usuário que está logado, precisamos também atualizar o **Shared Preferences**.



Flutter 2.0

Basicamente alterações de null safety

```
Future<bool> atualizar_usuario(BuildContext context) async{
  if(usuario != null){
    usuario.nome = controlador_nome.text;
    usuario.login = controlador_login.text;
    usuario.senha = controlador_senha.text;
    usuario.endereco = controlador_endereco.text;
    usuario.tipo = tipo_usuario_selecionado;
    if(imagem != null){
      // Se já havia foto pode ser necessário apagá-la
      if (usuario.urlFoto != null){
        // Se houve troca de foto
        if (imagem.path != usuario.urlFoto){
          GerenciadoraArquivo.excluirArquivo(usuario.urlFoto);
          usuario.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem);
        }
        // Se não havia foto é necessário salvá-la
      } else {
        usuario.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem);
      }
    }
    // Se o usuário logado é o que está sendo salvo
    // atualizamos as Shared Preferences
    if(usuario_logado.id == usuario.id) {
      usuario.salvar();
    }
    FabricaControladora.obterUsuarioControl().atualizarUsuario(usuario);
  }
}
```

```
Future<bool> atualizar_usuario(BuildContext context) async{
  if(usuario != null){
    usuario!.nome = controlador_nome.text;
    usuario!.login = controlador_login.text;
    usuario!.senha = controlador_senha.text;
    usuario!.endereco = controlador_endereco.text;
    usuario!.tipo = tipo_usuario_selecionado!;
    if(imagem != null){
      // Se já havia foto pode ser necessário apagá-la
      if (usuario!.urlFoto != null){
        // Se houve troca de foto
        if (imagem!.path != usuario!.urlFoto){
          GerenciadoraArquivo.excluirArquivo(usuario!.urlFoto!);
          usuario!.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem!);
        }
        // Se não havia foto é necessário salvá-la
      } else {
        usuario!.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem!);
      }
    }
    // Se o usuário logado é o que está sendo salvo
    // atualizamos as Shared Preferences
    if(usuario_logado.id == usuario!.id) {
      usuario!.salvar();
    }
    FabricaControladora.obterUsuarioControl().atualizarUsuario(usuario!);
  }
}
```



Persistindo a imagem

```
} else {  
  Usuario usuario_novo = Usuario(  
    nome: controlador_nome.text,  
    login: controlador_login.text,  
    senha: controlador_senha.text,  
    endereco: controlador_endereco.text,  
    tipo: tipo_usuario_selecionado,  
  );  
  if(imagem != null){  
    usuario_novo.urlFoto = await GerenciadoraArquivo.salvarImagem(imagem);  
  }  
  FabricaControladora.obterUsuarioControl().inserirUsuario(usuario_novo);  
}
```

- Podemos também estar trabalhando com um usuário novo.
- Nesse caso é mais simples, basta criar o usuário e salvar a imagem (caso exista).
Notar que o usuário armazena a **urlFoto** (que é o endereço da foto dentro do dispositivo). Essa **urlFoto** é importante para recuperar, em outro momento, a imagem salva.
- Por fim temos a inserção do usuário no banco de dados, assim como no slide anterior tivemos a atualização do usuário no banco de dados.



Obtendo a imagem persistida

- No método **inicializar** será necessário obter a imagem no caso de um usuário pré existente (não novo) e que tenha imagem.
- Note que a imagem é obtida a partir de **urlFoto**.

```
if(usuario != null) {  
    controlador_nome.text = usuario.nome;  
    controlador_login.text = usuario.login;  
    controlador_senha.text = usuario.senha;  
    controlador_endereco.text = usuario.endereco;  
    tipo_usuario_selecionado = usuario.tipo;  
    if (usuario.urlFoto != null){  
        imagem = await GerenciadoraArquivo.obterImagem(usuario.urlFoto);  
    }  
}
```



GerenciadoraArquivo

- A classe **GerenciadoraArquivo** é uma classe utilitária criada para persistir/gerenciar arquivos no dispositivo.
- Para seu funcionamento é necessário instalar o plugin **path_provider**, cuja instalação é similar a de outros plugins instalados e se encontra em:

https://pub.dev/packages/path_provider#-installing-tab-




```
import 'dart:async';
import 'dart:io';
import 'package:path_provider/path_provider.dart';

class GerenciadoraArquivo {

  static Future<String> get _LocalPath async {
    final directory = await getApplicationDocumentsDirectory();
    return directory.path;
  }
}
```

- A **GerenciadoraArquivo** precisa fazer a importação do arquivo **path_provider.dart**. Esse arquivo possui o método **getApplicationDocumentsDirectory** e através dele teremos acesso ao caminho do diretório do aplicativo (para que possamos armazenar e obter nossos arquivos no local correto).



```

static Future<String> salvarImagem(File arquivo) async{
    DateTime dateTime = DateTime.now();
    String nome_arquivo = dateTime.millisecondsSinceEpoch.toString();
    final path = await _localPath;
    await arquivo.copy('$path/$nome_arquivo.png');
    return '$path/$nome_arquivo.png';
}

static Future<File> obterImagem(String path)async {
    return File(path);
}

static void excluirArquivo(String path) async {
    File(path).delete(recursive: true);
}

```

- O nome do arquivo é dado pelo momento atual em milissegundos. O objetivo aqui é garantir um nome único para cada arquivo.
- O local de armazenamento é a pasta do aplicativo. Note que é feita uma cópia do arquivo em memória (gerado pelo image_picker) para a pasta do aplicativo com o nome gerado a partir do tempo em milissegundos.
- Para obter a imagem posteriormente basta passar o caminho para a classe **File**.
- Para excluir a ideia é basicamente a mesma.



Flutter 2.0

- Basicamente acrescentamos o método XFileToFile (para suporte de Foto e Galeria):

```
static File XFileToFile(XFile arquivoX){  
    return File(arquivoX.path);  
}
```



Dúvidas?

