RELATÓRIO

Trabalho Prático 2 - Flutter

Arquiteturas Móveis

Licenciatura em Engenharia Informática

Desenvolvimento de uma aplicação para apresentação e atualização da ementa semanal da cantina do ISEC, tirando partido de uma API.

Trabalho realizado por:

Tomás Gomes Silva - 2020143845

Rafael Gerardo Couto - 2019142454

Tânia Beatriz Moreira Guedes - 2020139445

Índice

Introdução

O segundo trabalho prático de Arquiteturas Móveis consiste na criação de uma aplicação, em Flutter, que seja capaz de tirar partido de uma API para mostrar o menu semanal da cantina do ISEC. Para além disso esta aplicação tem de ser possível editar a refeição caso o utilizador detete que esta se encontra errada.

Funcionalidades

Icon e animações

O icon da aplicação foi alterado através do Android Studio. Adicionalmente, visto que alguns dos membros do grupo possuía dispositivos Apple (como um iPhone e Mac), alterámos também o icon para estas plataformas.
Quanto às animações utilizámos o widget Hero para animar as transições entre a página inicial e o ecrã de edição e vice-versa. Os widgets animados foram o nome do dia da semana e a imagem do menu.

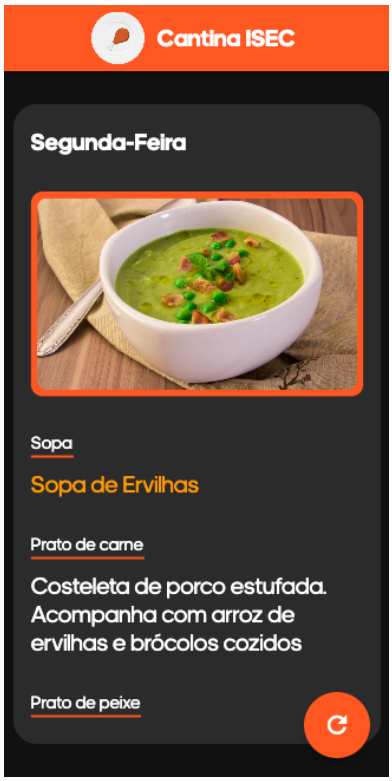
```
Hero(
  tag: weekDay,
  child: Material(
    color: Colors.transparent,
    child: Text(
      weekDay,
      style: const TextStyle(
        color: Colors.white,
        fontSize: 20,
        fontWeight: FontWeight.bold,
        fontFamily: 'AdiNeuePRO',
      ), // TextStyle
    ), // Text
  ), // Material
), // Hero
```

```
Hero(
  tag: weekDay,
  child: Material(
    color: Colors.transparent,
    child: Text(
      weekDay,
      style: const TextStyle(
        color: Colors.white,
        fontSize: 20,
        fontFamily: 'AdiNeuePRO',
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // Text
  ), // Material
), // Hero
```

O Hero funciona graças às tags que são dadas ao widget. Com base nestas tags, a transição é efetuada entre dois elementos que tenham a mesma tag. É, por este motivo, bastante importante que as tags sejam únicas.

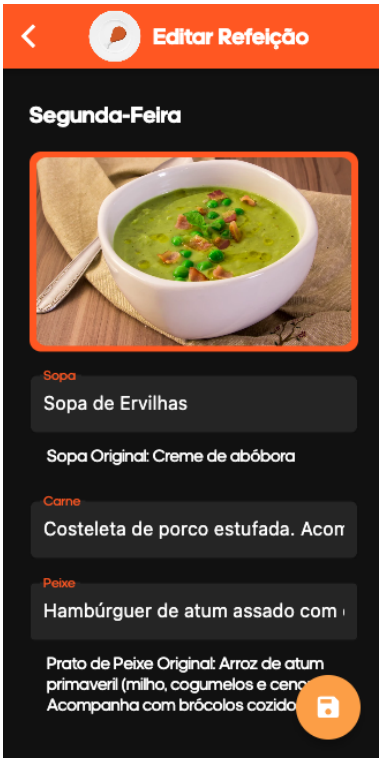
Ecrã principal

O ecrã principal da aplicação consiste em 5 widgets (MealCard) distribuídos numa ListView com scroll horizontal. Estes elementos são criados através da informação guardada nas shared-preferences ou através de um pedido à API facultada pelos docentes que retorna uma resposta em JSON que contém toda a informação relativa à ementa de uma semana.
Os campos a laranja indicam que a informação foi editada por utilizador e que esta não corresponde à informação original.
No canto inferior direito da página encontra-se um botão que permite ao utilizador atualizar as informações apresentadas pela aplicação.



Ecrã de edição

O ecrã de edição, como mencionado anteriormente permite a um utilizador efetuar a atualização das informações de uma ementa.
A página consiste na fotografia da ementa (que não pode ser editada) e em TextFormFields que permitem ao utilizador inserir a nova informação.
Estes campos vêm pré-preenchidos de acordo com a informação presente no menu inicial. Se um campo estiver vazio então a API trata de sobrescrever a informação do menu com a informação original.
Quando um campo de uma ementa não corresponde à informação original a aplicação mostra em baixo do campo o valor original da ementa.
Finalmente, existe também um FloatingActionButton que permite ao utilizador submeter as alterações efetuadas. Se esta operação for efetuada com sucesso, o utilizador é redirecionado para o ecrã principal e a informação presente nos widgets é atualizada.



Utilização da API

Para tirar partido da API utilizámos o plugin http que nos permitiu fazer pedidos GET e POST à API que por sua vez retornava uma resposta a um pedido.
Utilizámos o método GET na página principal para atualizar as shared preferences e na página de edição para obter a imagem do menu em base64.

```
Future<http.Response> response = http.get(Uri.parse('http://94.61.156.105:8080/menu'));
response.then((value) {
  SharedPreferences.getInstance().then((prefs) {
    prefs.setString('weeklyMenu', value.body);
  });
  menu = jsonDecode(value.body);
```

```
if (filePath != "images/emptyMenu.png") {
  var response = await http.get(Uri.parse(filePath));
  var bytes = response.bodyBytes;
  base64Img = base64Encode(bytes);
}
```

Já no método POST tivemos de definir um cabeçalho para enviar no pedido e preenchemos um JSON com a informação necessária para atualizar o menu.

```
Future<http.Response> response = http.post(Uri.parse('http://94.61.156.105:8080/menu'),
    body: jsonEncode({
      'weekDay': diaDaSemana,
      'soup': _soupController.text,
      'fish': _fishController.text,
      'meat': _meatController.text,
      'vegetarian': _vegetarianController.text,
      'desert': _dessertController.text,
      'img': base64Img,
    }),
    headers: {'Content-Type': 'application/json; charset=UTF-8'});
response.then((value) async => {
```

Persistência de ementas

Para que o utilizador possa ter acesso à ementa offline, utilizámos o plugin shared-preferences para que sempre que o utilizador carregue no botão de atualizar o menu este seja guardado nas Shared Preferences do dispositivo. Ao iniciar a aplicação é carregado o menu a partir daí e os widgets são preenchidos com essa informação.
O plugin dart-convert também foi muito utilizado para efetuar conversões de strings para objetos JSON. Para além disso utilizámos este mesmo plugin para converter uma imagem para base64, como irá ser mostrado mais à frente neste relatório.

```
SharedPreferences.getInstance().then((prefs) {
  prefs.setString('weeklyMenu', value.body);
});
```

```
if (prefs.getString('weeklyMenu') != null) {
  setState(() {
    menu = jsonDecode(prefs.getString('weeklyMenu')!);
  });
```

Apresentação das imagens dos menus

Em relação à apresentação das imagens dos menus, de modo a sabermos que imagem apresentar recorremos a operadores ternários de modo a identificar a imagem de acordo com o dia da semana. Na figura a baixo apresentamos uma parcialidade do código relativo à criação da MealCard que compõem as todas as informações para cada dia da semana e imagem.

```
MealCard(
  tag: i.toString(),
  weekDay: days[i],
  imagePath: i == 0
      ? imageMonday
      : i == 1
          ? imageTuesday
          : i == 2
              ? imageWednesday
              : i == 3
                  ? imageThursday
                  : imageFriday,
```

Uma das decisões de implementação relativamente à apresentação das imagens dos menus passou por apresentar uma imagem default caso não haja imagem para o dia da semana em questão. Na figura a baixo pode ser visto um excerto de código representativo do modo de obtenção do path da imagem caso exista, novamente recorrendo a operadores ternários.

```
imageMonday = menu["MONDAY"]["update"] != null
    ? menu["MONDAY"]["update"]["img"] != ''
        ? "http://94.61.156.105:8080/images/${menu["MONDAY"]["update"]["img"]}"
        : "images/emptyMenu.png"
    : "images/emptyMenu.png";
```

Relativamente à construção do próprio Widget que suporta a imagem, há que fazer igualmente a distinção se há imagem ou não, pois se não existir é um Image.asset visto que a imagem está local, caso contrário terá de ser um Image.network para que seja possível carregar a imagem através da rede.

```
if (imagePath == 'images/emptyMenu.png') {
  return Image.asset(
    imagePath,
    fit: BoxFit.contain,
    height: screenHeight * 0.25,
  ); // Image.asset
} else {
  return Image.network(
    imagePath,
    fit: BoxFit.contain,
    height: screenHeight * 0.25,
  ); // Image.network
```