



Aula 6 – Acesso a Dados

Desenvolvimento de aplicações híbridas com Flutter

05-06 de Novembro/21

Agenda

- Armazenar dados Chave-Valor.
- Trabalhando com arquivos.
- Banco de dados Relacional (SQLite)
- Networking
- Desafio

Armazenando dados Chave-Valor

Armazenando dados Chave-Valor

É possível utilizar ferramentas nativas do sistema para armazenar dados em formato Chave-Valor.

Android:

SharedPreferences

iOS:

NSUserDefaults

Armazenando dados Chave-Valor

Por sua limitação, normalmente se armazena dados com estruturas simples.

Prós:

- Facilidade de utilização.
- Leve.

Contras:

- Dificuldade de versionamento.
- Pouco flexível.

Implementação em Flutter

Para utilizar os recursos de armazenamento de dados Chave-Valor, é necessário utilizar o plugin **shared_preferences**:

1. Declarar a dependência do plugin no arquivo *pubspec.yaml*

```
dependencies:  
  flutter:  
    sdk: flutter  
  
  # The following adds the Cupertino Icons font to your application.  
  # Use with the CupertinoIcons class for iOS style icons.  
  cupertino_icons: ^1.0.2  
  shared_preferences: ^2.0.7  
  path_provider: ^2.0.4
```

Implementação em Flutter

Dica:

```
$ flutter pub add shared_preferences
```

Adiciona a dependência 'shared_preferences' no projeto.

```
$ flutter pub get
```

Baixa as dependências necessárias, de acordo com o arquivo *pubspec.yaml*

Armazenando dados Chave-Valor

Adicionar novos dados

```
// obtain shared preferences
final prefs = await SharedPreferences.getInstance();

prefs.setInt('counter', myInt);
...
prefs.setDouble('double_counter', myDouble);
...
prefs.setString('name', myString);
...
prefs.setStringList('names', myStringList);
...
prefs.setBool('config', myBool);
```


Armazenando dados Chave-Valor

Ler dados

```
// obtain shared preferences
final prefs = await SharedPreferences.getInstance();

prefs.getInt('counter') ?? 0;
...
prefs.getDouble('double_counter') ?? 0.0;
...
prefs.getString('name') ?? "";
...
prefs.getStringList('names') ?? [];
...
prefs.getBool('config') ?? false;
```

Armazenando dados Chave-Valor

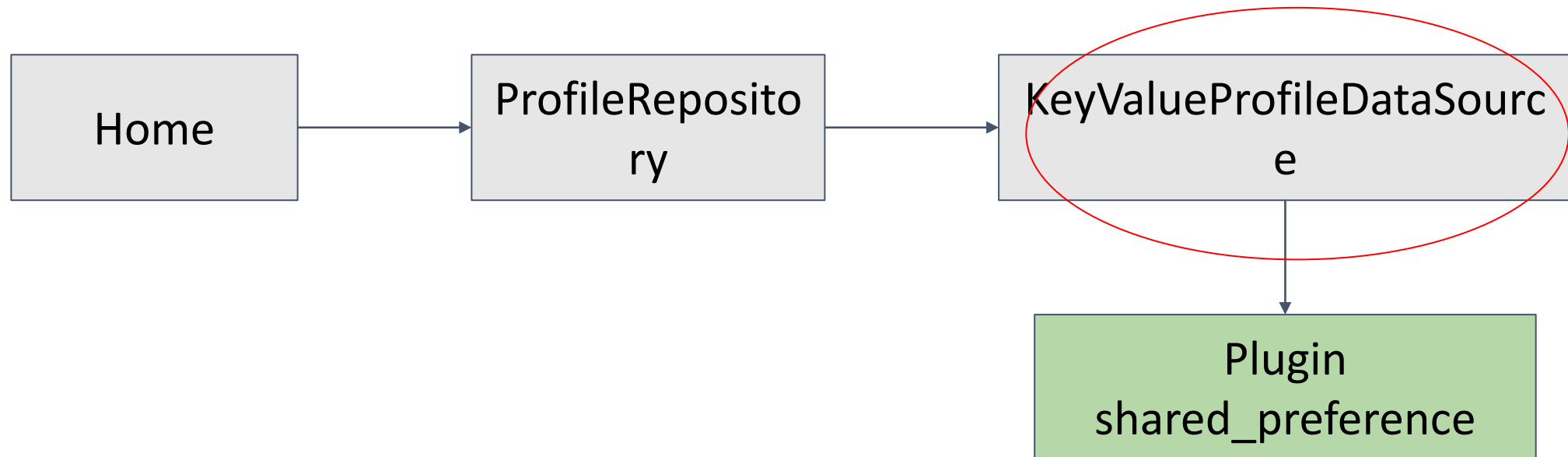
Deletar dados

```
// obtain shared preferences  
final prefs = await SharedPreferences.getInstance();  
  
prefs.remove('counter');
```

Armazenando dados Chave-Valor

Atividade 1

Baixar o código do projeto no github e implementar um DataSource que utilize Shared Preferences para armazenar as informações do Perfil



Trabalhando com arquivos

Trabalhando com arquivos

O dart possui uma biblioteca de acesso à arquivo chamada [dart:io](#). Entretanto, precisamos resolver o onde o app vai armazenar os arquivos.

Trabalhando com arquivos

Decidindo o diretório ideal

O pacote `path_provider` resolve os diretórios específicos dos sistemas para dois tipos de diretórios :

- *Diretório temporário*
- *Diretório de documentos*

Trabalhando com arquivos

Diretório temporário

Para arquivos temporários, onde não há garantia que o dado persistirá para sempre.

Android:

`getCacheDir()`

iOS:

`NSCachesDirectory`

Trabalhando com arquivos

Diretório de Documentos

Para arquivos exclusivos ao App. É excluído apenas quando o app é removido ou tem os dados eliminados.

Android:

App Data

IOS:

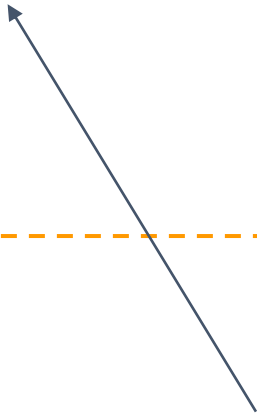
NSDocumentDirectory

Trabalhando com arquivos

Obtendo o Diretório de documentos com Flutter

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

Função da biblioteca
`path_provider`



Trabalhando com arquivos

Criando uma referencia de arquivo Dart

```
Future<File> get _localFile async {  
  final path = await _localPath;  
  return File('$path/arquivo.txt');  
}
```

Trabalhando com arquivos

Lendo de um arquivo

```
Future<String?> readText() async {  
  try {  
    final file = await _localFile;  
  
    final contents = await file.readAsString();  
  
    return contents;  
  } catch (e) {  
    return null;  
  }  
}
```

Trabalhando com arquivos

Escrevendo em um arquivo

```
Future<File> writeCounter(int counter) async {  
    final file = await _localFile;  
  
    // Write the file  
    return file.writeAsString('$counter', mode: FileMode.write);  
}
```

Modos de escrita:

FileMode.write

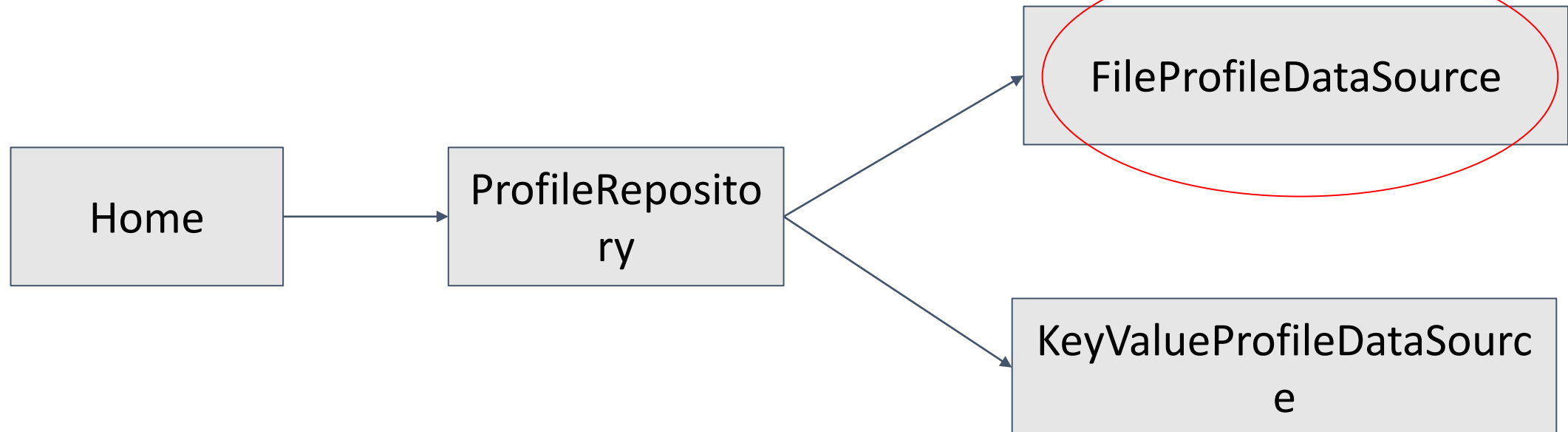
FileMode.read

FileMode.append

Armazenando dados Chave-Valor

Atividade 2

Baixar o código do projeto no github e implementar um DataSource que utilize arquivos para armazenar as informações do Perfil



Banco de dados Relacional (SQLite)

Banco de dados relacional

É possível trabalhar com banco de dados relacional utilizando a biblioteca [sqlite](#).

- Fácil utilização
 - Flutter abstrai maior parte do gerenciamento do banco de dados.
- Pouco overhead
 - Por trabalhar diretamente com arquivos, SQLite é ideal para utilização em dispositivos móveis.

Banco de dados relacional

Definindo um modelo

```
class Profile {  
  int? id;  
  String name;  
  String address;  
  
  Profile({  
    this.id,  
    required this.name,  
    required this.address,  
  });  
}
```


Banco de dados relacional

Iniciando o banco de dados

```
final database = openDatabase(  
  join(await getDatabasesPath(), 'profile.db'),  
);
```

Garante que o path estará
correto em qualquer
Sistema Operacional

Banco de dados relacional

Definindo tarefas na primeira inicialização

```
final database = openDatabase(  
  join(await getDatabasesPath(), 'profile.db'),  
  onCreate: (db, version) {  
    return db.execute(  
      'CREATE TABLE PROFILE(id INTEGER PRIMARY KEY, name TEXT, address TEXT)',  
    );  
  },  
  version: 1,  
);
```

Executará quando o banco
for criado pela primeira vez.

Banco de dados relacional

Inserindo um item na tabela

Para poder inserir o item no banco, primeiro é necessário converter o objeto para um Map:

```
class Profile {  
    ...  
    Map<String, dynamic> toMap() {  
    return {  
        'id': id,  
        'name': name,  
        'address': address,  
    };  
    }  
}
```

Banco de dados relacional

Inserindo um item na tabela

```
Future<void> setProfile(Profile profile) async {  
    final db = await database;  
  
    await db.insert(  
        'profile',  
        profile.toMap(),  
        conflictAlgorithm: ConflictAlgorithm.replace,  
    );  
}
```

Banco de dados relacional

Consultando o banco

```
Future<Profile> getProfile() async {  
  final db = await database;  
  
  final List<Map<String, dynamic>> maps = await db.query('profile');  
  
  return Profile(  
    id: maps[0]['id'],  
    name: maps[0]['name'],  
    address: maps[0]['address'],  
  );  
}
```

Banco de dados relacional

Atualizando o banco

```
Future<void> updateProfile(Profile profile) async {  
    final db = await database;  
  
    await db.update(  
        'profile',  
        profile.toMap(),  
        where: 'id = ?',  
        whereArgs: [profile.id],  
    );  
}
```

Banco de dados relacional

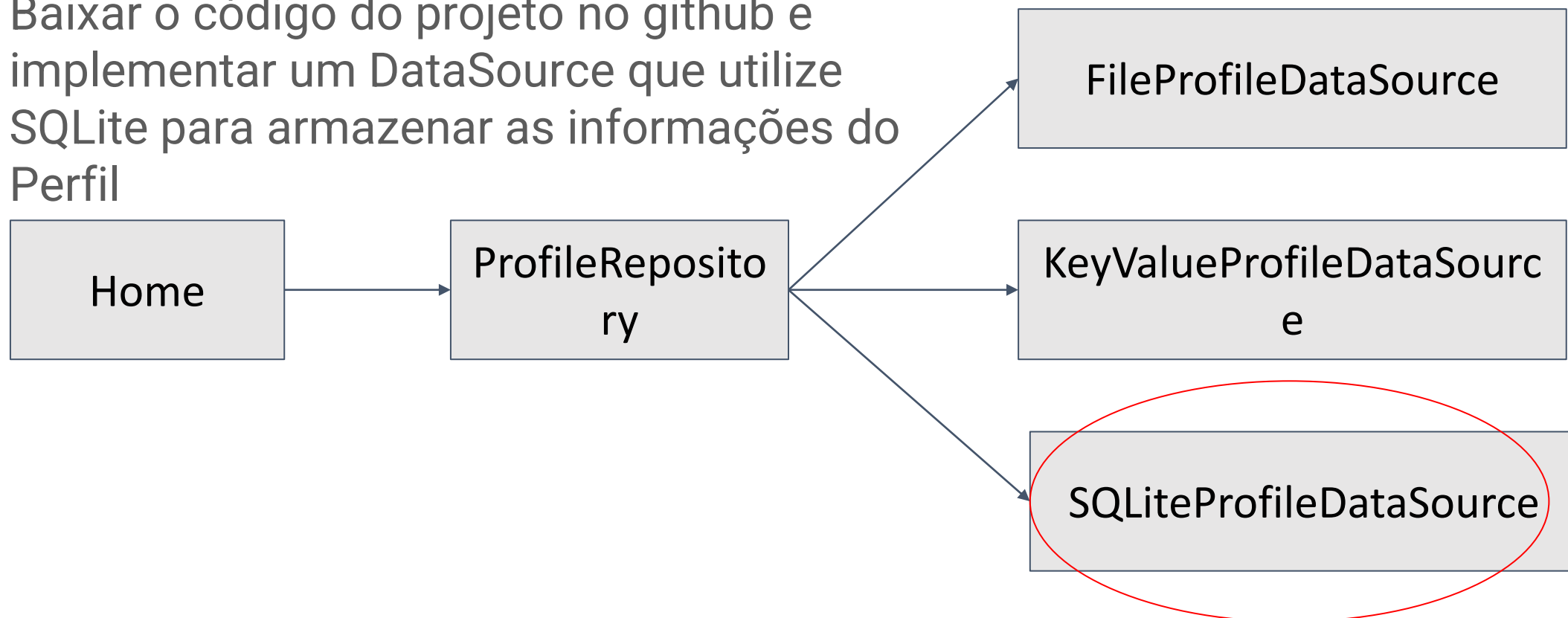
Removendo um item do banco

```
Future<void> updateProfile(Profile profile) async {  
    final db = await database;  
  
    await db.delete('profile', <Parâmetros da query>);  
}
```

Banco de dados relacional

Atividade 3

Baixar o código do projeto no github e implementar um DataSource que utilize SQLite para armazenar as informações do Perfil



Networking

Networking

Introdução

Flutter provê acesso ao protocolo http por meio da biblioteca [http](#)

Além disso, é necessário declarar a permissão de internet no AndroidManifest.xml.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Networking

Requisição básica

```
Future<http.Response> fetchAlbum() {  
    return http.get(Uri.parse('https://jsonplaceholder.typicode.com/albums/1'));  
}
```

O objeto `http.Response` possui informações sobre a resposta do servidor, como o `body`, `statusCode`, `headers` etc.

Networking

Converter resposta para objeto modelo

O Flutter não possui um mecanismo para mapeamento de Json para Objeto, sendo necessário fazer manualmente

```
class Profile {  
    ...  
    factory Profile.fromJson(Map<String, dynamic> json) {  
        return Profile(  
            id: json[id],  
            name: json['name'],  
            address: json['address'],  
        );  
    }  
}
```

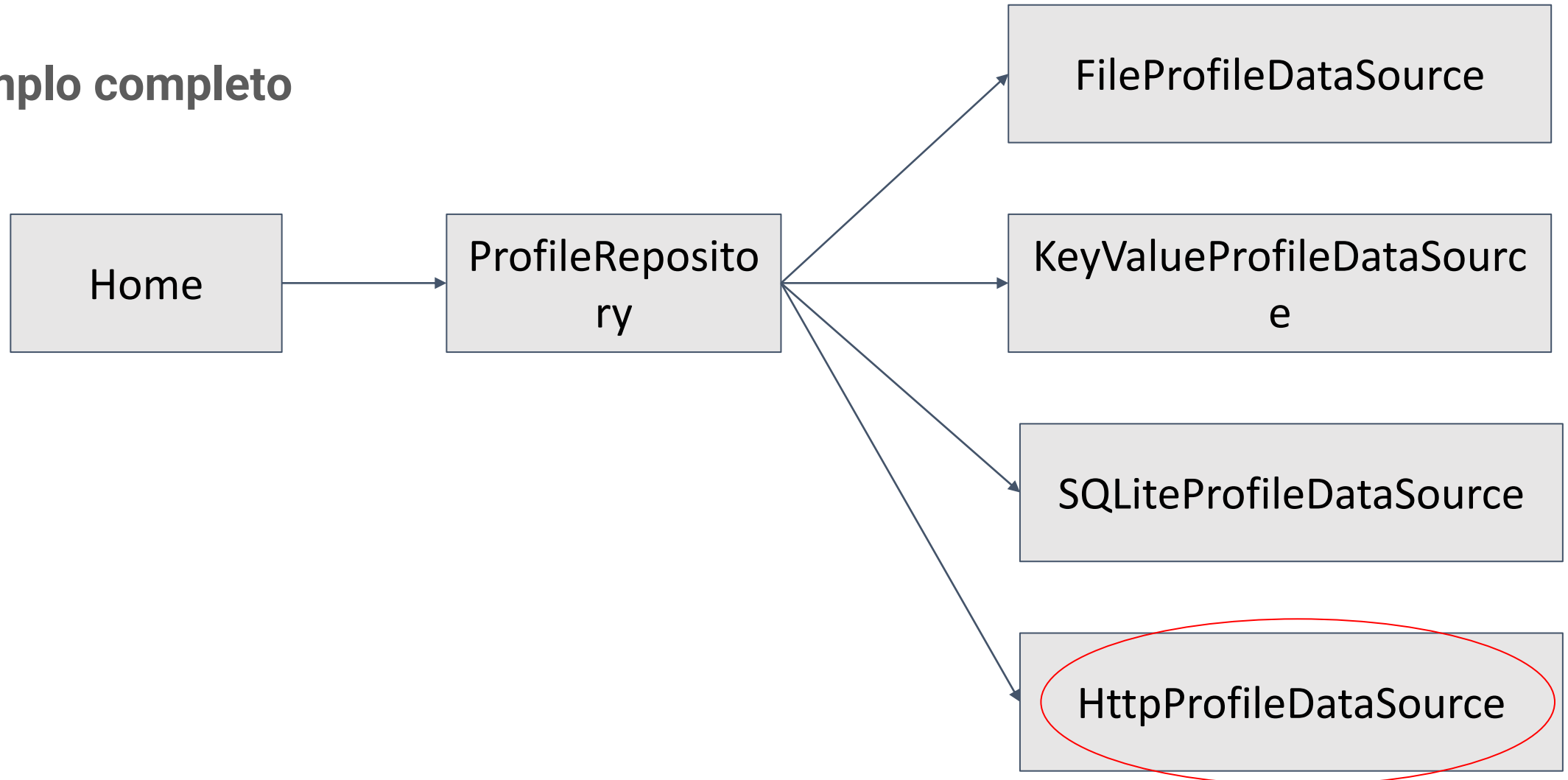
Networking

Enviando dados para o Servidor

```
final response = await http.post(  
  Uri.parse('https://jsonplaceholder.typicode.com/albums'),  
  headers: <String, String>{  
    'Content-Type': 'application/json; charset=UTF-8',  
  },  
  body: jsonEncode(<String, String>{  
    'title': title,  
  })),  
);
```

Networking

Exemplo completo



Desafio 2

<https://drive.google.com/file/d/1b0cD2Qv1tV0C6Gwro928bE-d7qsGMNil/view?usp=sharing>



Obrigado!

05-06 de Novembro/21