Departamento de Engenharia Informática
TeSP em Programação de Sistemas de Informação

Acesso Móvel a Sistemas de Informação
2º Ano – 1º Semestre
2021/2022

Docentes: Sónia Luz, sonia.luz@ipleiria.pt
David Safadinho, david.safadinho@ipleiria.pt
Cátia Ledesma, catia.ledesma@ipleiria.pt

Ficha Prática 9
Books – Acesso a API REST

Objetivos da Ficha

- Acesso a uma API REST.

Introdução

Nesta ficha vamos continuar a aplicação Books mas com acesso a dados dinâmicos a partir de uma API REST, para permitir o acesso e persistência dos dados da aplicação através da internet.

Abrir a pasta do projeto da aula anterior

1. Neste projeto deve passar a considerar uma API REST para persistir os dados através da internet.
   1.1. Iremos passar a utilizar duas bibliotecas disponibilizadas pelo Android, nomeadamente **Volley** (para acesso à API REST) e **Glide** (para acesso e representação de imagens através de url);
      1.1.1. Para isso deve adicionar as bibliotecas ao ficheiro **build.gradle** através das seguintes linhas de código:

```
dependencies {
...
    implementation 'com.android.volley:volley:1.2.1'
    implementation 'com.github.bumptech.glide:glide:4.10.0'
}
```

      1.1.2. Devendo de seguida fazer a sincronização do projeto;

1.1.3. Após isso, de modo a conseguir aceder à API, deve alterar o ficheiro **AndroidManifest.xml** para adicionar permissões de acesso a internet, acesso ao estado da rede, e com as seguintes linhas de código:

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

<application
    ...
     android:usesCleartextTraffic="true">
```

1.2. Para preparar a implementação desta ficha terá de fazer algumas alterações às classes existentes, com o intuito de deixar de trabalhar apenas com imagens de capas de livros locais;

1.2.1. Na classe **Livro:**

1.2.1.1. O construtor do livro passa a receber o **id** por parâmetro para iniciar o atributo e deve eliminar o atributo ***autoIncrementId***, pois deixa de ser necessário;

1.2.1.2. Deve corrigir todas as instâncias de Livro criadas.

1.2.1.3. A capa do livro deve passar a ser do tipo **String**, para guardar o url da imagem. Depois deve alterar os respetivos métodos assessores e corrigir qualquer uso do respetivo atributo, não esquecendo de alterar o tipo de dados utilizado na criação da tabela na BD local;

1.2.2. Na classe **LivroBDHelper** deve alterar o método onCreate, onde é feita a criação da tabela livros.

1.2.3. Considerando que agora guardamos o **url** da imagem, temos de alterar as classes **ListaLivroAdaptador** e **GrelhaLivroAdaptador**, de modo a conseguirem aceder e apresentar a respetiva imagem;

1.2.3.1. No método **update()** para atribuir a imagem à **capa** do livro deve utilizar a biblioteca Glide com o seguinte código:

```
Glide.with(context)
        .load(livro.getCapa())
        .placeholder(R.drawable.logoipl)
        .diskCacheStrategy(DiskCacheStrategy.ALL)
        .into(imgCapa);
```

1.2.4. Devido à alteração do tipo de dados da capa do livro, deve alterar a classe **LivroBDHelper**.

1.2.5. Também devido à alteração do tipo de dados da capa do livro, deve modificar a classe **DetalhesLivroActivity**;

1.2.5.1. Assim, quando cria novo livro, deve considerar um **url** por omissão para atribuir à imagem da capa, por exemplo "**http://amsi.dei.estg.ipleiria.pt/img/ipl_semfundo.png**",

1.2.5.2. Ao criar um novo livro deve atribuir ao **id** o valor **zero**, que posteriormente será atualizado através da API;

1.2.5.3. Para preencher a capa do livro a ser representado, deve recorrer novamente à biblioteca **Glide**;

1.3. Após isso, deve criar o package **utils**;

1.4. No package **utils**, deve criar uma classe **LivroJsonParser** para fazer o mapeamento entre os dados que vêm da API e o nosso modelo. Assim deve implementar os seguintes métodos estáticos de *parsing* entre a classe **Livro** e o objeto Json;

1.4.1. Método **parserJsonLivros()**, que devolve a lista de livros;

1.4.2. Método **parserJsonLivro()**, que devolve apenas um livro;

1.4.3. Método **parserJsonLogin()**, que efetuará o login na API;

1.4.4. Método **isConnectionInternet()**, que verifica se existe acesso à internet;

1.5. Na classe **SingletonGestorLivros**, vamos efetuar diversas alterações:

1.5.1. É necessário criar um atributo estático do tipo **RequestQueue volleyQueue** e instanciá-lo no método **getInstance()**;

1.5.2. É necessário criar um atributo do tipo **String**, **mUrlAPILivros**, com o url de acesso aos livros da API: "**http://amsi.dei.estg.ipleiria.pt/api/livros**";

1.5.3. Para garantir que os dados dos livros da BD são atualizados com os dados vindos da API, deve implementar o método **adicionarLivrosBD()** na classe **SingletonGestorLivros**, e o método **removerAllLivrosBD()** na classe **LivroBDHelper**, que tem como intuito eliminar todos os livros da BD.

1.5.3.1. Deve ainda fazer uns ajustes aos métodos que fazem pedidos à BD.

1.5.4. Para os pedidos adicionar livro e editar livro efetuados à API é necessário um token, para obtê-lo aceda ao link: http://amsi.dei.estg.ipleiria.pt/api

1.5.5. Agora deve adicionar os métodos para aceder à API:

1.5.5.1. Método para adicionar um livro – **void adicionarLivroAPI(final Livro livro, final Context context)**, que irá necessitar do token para efetuar o pedido;

1.5.5.2. Método para aceder a todos os livros oriundos da API – **void getAllLivrosAPI(final Context context)**;

1.5.5.3. Método para remover um livro – **void removerLivroAPI(final Livro livro, final Context context)**;

1.5.5.4. Método para editar os dados de um livro – **void editarLivroAPI(final Livro livro, final Context context)**, que irá necessitar do token para efetuar o pedido;

1.6. Quando se efetuam pedidos à API pode suceder a aplicação apresentar os resultados antes de obter a resposta do servidor. Por esse motivo será necessário implementar um objeto ***Listener***, que irá desempenhar o papel de ouvinte à escuta de atualizações na lista de livros;

1.6.1. Assim, deve criar o package **listeners**;

1.6.2. No package **listeners** deve criar uma interface **LivrosListener** com o método **void onRefreshListaLivros(ArrayList<Livro> listaLivros)** que irá ser posteriormente implementado pelas classes que implementarem o referido **listener**;

1.6.2.1. Para saber qual o **listener** ativo deve adicionar na classe **SingletonGestorLivros** um atributo do tipo **LivrosListener** e respetivo método *setter*;

1.6.3. As classes **ListaLivrosFragment** e **GrelhaLivrosFragment** devem implementar a interface **LivrosListener**;

1.6.3.1. Após isso será necessário redefinir o método **onRefreshListaLivros()**;

1.6.3.2. No método **onCreateView()** destes fragmentos é necessário registar cada classe como **listener** do **SingletonGestorLivros** e de seguida obter os dados de todos os livros existentes na API;

1.6.3.3. Nestas classes deixará de ser usado o atributo referente à lista de livros, pois vai-se aceder aos métodos do Singleton para obter os dados atualizados;

1.6.4. Na classe **DetalhesLivroActivity** deve passar a aceder aos métodos que manipulam os dados diretamente na API;

1.6.4.1. Crie uma nova interface denominada **LivroListener** com o método **void onRefreshDetalhes(int op)** que irá ser implementado pela **DetalhesLivroActivity**.

1.6.4.2. Os métodos **adicionarLivroBD**, **removerLivroBD** e **editarLivroBD** passam a ser substituídos pelos métodos criados atrás: **adicionarLivroAPI, removerLivroAPI** e **editarLivroAPI.**

1.6.5. De forma a permitir um acesso restrito à API, devemos acrescentar a validação de dados para que o email do utilizador e respetiva password sejam verificados e quando corretamente inseridos, sendo devolvido o respetivo **token** de acesso;

1.6.5.1. Assim, na classe **SingletonGestorLivros** deve:

1.6.5.1.1. Criar um atributo do tipo **String**, **mUrlAPILogin**, com o url de acesso ao login da API: "**http://amsi.dei.estg.ipleiria.pt/api/auth/login**"

1.6.5.1.2. Implementar o método **loginAPI(final String email, final String password)** para efetuar o pedido de acesso à API;

1.6.5.2. No package **listeners** deve criar uma interface **LoginListener** com o método **void onValidateLogin(final String token, final String email, final Context context)** que irá ser posteriormente implementado pelas classes que implementarem o respetivo **listener**;

1.6.5.3. Para saber qual o **listener** ativo, deve adicionar na classe **SingletonGestorLivros** um atributo do tipo **LoginListener** e respetivo método *setter*;

1.6.5.4. A classe **LoginActitivity** deve implementar a interface **LoginListener**;

1.6.5.4.1. Será necessário redefinir o método **onValidateLogin ();**

1.6.5.4.1.1. Nesse método deve iniciar a atividade **MenuMainActivity** enviando o token e o email para ser armazenado no Shared Preferences.

1.6.5.4.2. No método **onCreate()** da classe **LoginActitivity** será necessário registá-la como **listener** do **SingletonGestorLivros**;

1.6.5.5. Na classe **SingletonGestorLivros** será necessário alterar os métodos **adicionarLivroAPI e editarLivroAPI,** devendo estes passar a receber o **tokenAPI** como parâmetro, que está armazenado no Shared Preferences.

1.6.5.6. Para efetuar o login na API deve ir à classe **LoginActivity**, e no método **onClickLogin()** invocar o método **loginAPI()** da classe **SingletonGestorLivros**, passando-lhe o email e a password validados.

Observação: Para se poder autenticar terá de ter credenciais válidas, devendo aceder ao link http://amsi.dei.estg.ipleiria.pt/api para as obter.