Desenvolvimento para Dispositivos Móveis

Prof. Dr. Alan Souza

alan.souza@unama.br

2020

10. Consumindo API

- Normalmente, os dados apresentados em um app estão em um servidor e podem ser acessados através da internet;
- O acesso se dá através de uma API (*Application Programming Interface* - Interface de Programação de Aplicativos);
- Uma API é capaz de ligar aplicações diferentes através dos dados. Utiliza-se uma URL que gera os dados em JSON (pode ser outros formatos também, i. e., XML, HTML, Texto/Plano);
- A cada dia, cresce o número de APIs, pois a geração de dados também aumenta.

10. Consumindo API

- O objetivo dessa aula é criar um aplicativo que mostra previsão de dados climáticos;
- Iremos usar:
  - API OpenWeather: dados
  - RecyclerView: lista
  - Jsoup: framework para comunicação com a API via internet
  - Classes JSONObject, JSONArray: conversão dos dados da API em objetos do Java

10. Consumindo API

- Passos:

1. Entender um pouco como funciona a API OpenWeather;
2. Criar o projeto de acordo com as aulas anteriores;
3. Adicionar o RecyclerView
4. Adicionar o layout de um item do RecyclerView
5. Atribuir permissão de acesso à Internet no aquivo AndroidManifest.xml
6. Criar a classe Java referente aos dados que vai receber o resultado da API
7. Programar a comunicação com a API (Service com Jsoup)
8. Testar a aplicação
9. Adicionar melhorias

10. Consumindo API

- API OpenWeather:
  - Mostra uma série de informações do clima de várias cidades ao redor do mundo, inclusive a previsão de vários dias para frente, mas de 3 em 3 horas;
  - É possível criar uma conta gratuita, mas limitada;
  - É necessário realizar cadastro no site (link abaixo), para obter acesso à API *key*;
  - Link de cadastro: https://openweathermap.org/home/sign_up
  - Link da documentação da API: https://openweathermap.org/api

10. Consumindo API

- API OpenWeather:
  - URL que vamos usar para conexão:

api.openweathermap.org/data/2.5/forecast?

q=Belem,br **// cidade e país**

&appid=*api_key* **// pegar depois que efetua o cadastro**

&units=metric **// temperatura em ºC**

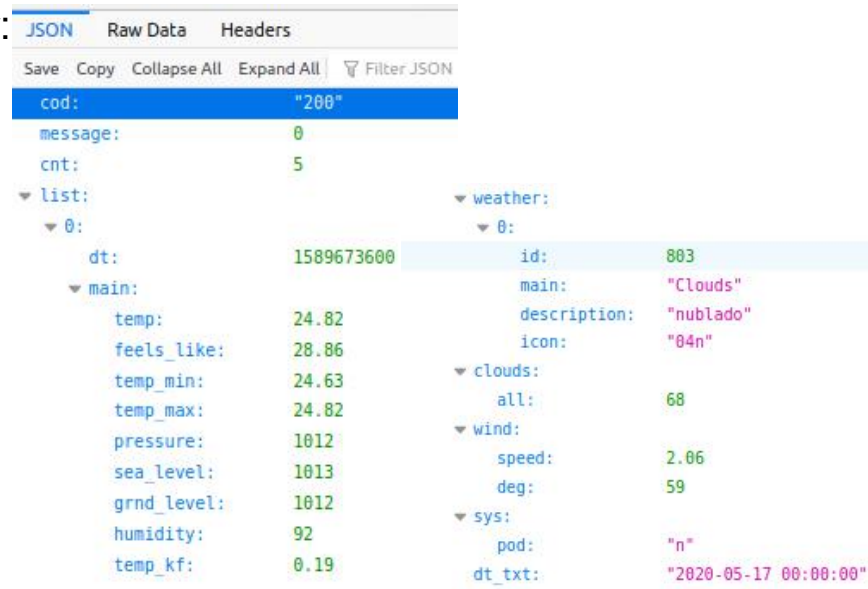
&lang=pt_br **// idioma do resultado**

&cnt=5 **// quantidade de registros**

OBS: É possível usar outros parâmetros, tal como coordenadas de latitude e longitude.

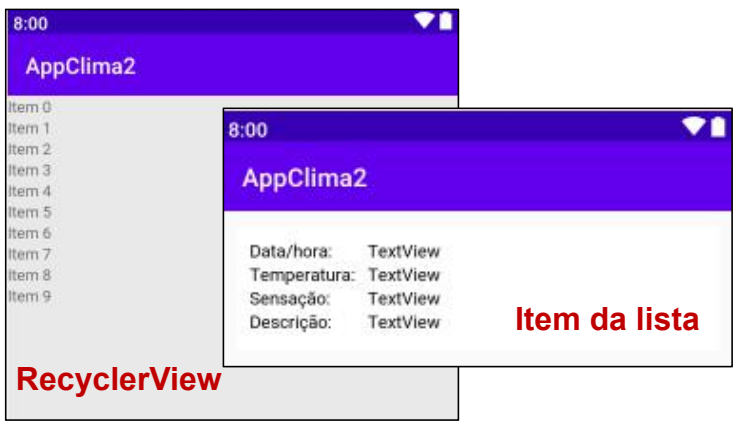
10. Consumindo API

- API OpenWeather:
  - Exemplo de resultado:



10. Consumindo API

- Layout do Projeto:



10. Consumindo API

• Permissão de acesso à rede no **AndroidManifest.xml**

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.previsaoclimabelem">
    <uses-permission
android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <application ...
    </application>
</manifest>
```

10. Consumindo API

• Declarar a classe Clima:

```
public class Clima {
    private String data;
    private double temperatura;
    private double sensacaoTermica;
    private String descricao;
    // criar setters e getters...
}
```

10. Consumindo API

• Declarar a classe Utils:

```
public class Utils {
   public static String converterData(String dataUTC) {
      String dataConvertida = "";
      try {
         SimpleDateFormat df = new SimpleDateFormat("yyyy-MM-dd
HH:mm:ss", Locale.ENGLISH);
         df.setTimeZone(TimeZone.getTimeZone("UTC"));
         Date date = df.parse(dataUTC);
         df.setTimeZone(TimeZone.getDefault());
         // continua...
```

10. Consumindo API

• Declarar a classe Utils:

```
         // código anterior
         df = new SimpleDateFormat("dd/MM/yyyy HH:mm",
Locale.ENGLISH);
         dataConvertida = df.format(date);
      } catch (Exception e) {
         dataConvertida = dataUTC + " (UTC)";
      }
      return dataConvertida;
   } // fim do converterData
```

10. Consumindo API

- Declarar a classe Utils:

```
public static boolean isOnline(Context context) {
    ConnectivityManager cm = (ConnectivityManager)
context.getSystemService( Context.CONNECTIVITY_SERVICE );
    NetworkInfo netInfo = cm.getActiveNetworkInfo();
    if (netInfo == null) return false;
    if (!netInfo.isConnected()) return false;
    if (!netInfo.isAvailable()) return false;
    return true;
  } // fim do isOnline
} // fim da classe Utils
```

10. Consumindo API

- Usando o Jsoup;
- Colocar no build.gradle:

```
implementation 'org.jsoup:jsoup:1.13.1'
```

- Criar a interface AsyncDelegate:

```
public interface AsyncTaskDelegate {
    void processFinish(Object output);
}
```

- Criar a classe ClimaService herdando de AsyncTask 1/11:

```
public class ClimaService extends AsyncTask<String, Void,
List<Clima>> { ... }
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 2/11:

```
private AsyncTaskDelegate delegate = null;
private Context context;
public ClimaService(Context context, AsyncTaskDelegate responder){
    this.context  = context;
    this.delegate = responder;
} // fim do construtor
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 3/11:

```
protected void onPreExecute() {
    super.onPreExecute();
    // no final do projeto, vamos fazer uma melhoria aqui para criar um sinal de “carregando” enquanto o app consulta os dados na API
} // fim do onPreExecute
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 4/11:

```
@Override
protected List<Clima> doInBackground(String... params) {
try {
   String local  = params[0];
   String appid = params[1];
   String qtd    = params[2];
   String urlAPI = "https://api.openweathermap.org/data/2.5/forecast?q="
+ local + "&appid=" + appid + "&cnt=" + qtd +
"&units=metric&lang=pt_br";
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 5/11:

```
String dadosAPI = Jsoup.connect(urlAPI).
ignoreContentType(true).execute().body();
JSONObject jsonAPI = new JSONObject(dadosAPI);
JSONArray vetorClimaAPI = jsonAPI.getJSONArray("list");
List<Clima> listaClima = new ArrayList<>();
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 6/11:

```
for (int i = 0; i < vetorClimaAPI.length(); i++) {
  Clima c = new Clima();
  try {
    JSONObject objExterno = vetorClimaAPI.getJSONObject(i);
    String data = objExterno.getString("dt_txt");
    c.setData(data);
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 7/11:

```
JSONObject objetoMain = objExterno.getJSONObject("main");

c.setTemperatura( objetoMain.getDouble("temp") );
c.setSensacaoTermica( objetoMain.getDouble("feels_like") );
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 8/11:

```
JSONArray vetorWeatherAPI =
objExterno.getJSONArray("weather");
for (int k = 0; k < vetorWeatherAPI.length(); k++) {
  JSONObject objetoWeather = vetorWeatherAPI.getJSONObject(k);
  c.setDescricao( objetoWeather.getString("description") );
}
listaClima.add( c );
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 9/11:

```
} catch (JSONException e) {
  e.printStackTrace();
  Toast.makeText(context, "Erro no parse do vetor de
dados!\nDetalhes: "+e.getMessage(),
  Toast.LENGTH_SHORT).show();
} // fim do try-catch de leitura dos dados
} // fim do for da leitura de dados
return listaClima;
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 10/11:

```
    } catch (Exception e) {
      e.printStackTrace();
      return null;
    } // fim do try-catch do método doInBackground

    } // fim do método doInBackground
```

10. Consumindo API

- Usando o Jsoup;
- Continuando a classe ClimaService 11/11:

```
    @Override
    protected void onPostExecute(List<Clima> climas) {
      super.onPostExecute(climas);
      if(delegate != null) {
        delegate.processFinish(climas);
      }
    } // fim do método onPostExecute

} // fim da classe
```

10. Consumindo API

• ClimaAdapter (para RecyclerView) 1/5:

```
public class ClimaAdapter extends
        RecyclerView.Adapter<ClimaAdapter.ViewHolder> {
    List<Clima> climas;
    public ClimaAdapter(List<Clima> dados) {
        this.climas = dados;
    } // fim do construtor
    // continua...
```

10. Consumindo API

• ClimaAdapter (para RecyclerView) 2/5:

```
    public class ViewHolder extends RecyclerView.ViewHolder {
        public TextView txtData, txtTemp, txtSensacao, txtDescricao;
        public ViewHolder(View itemView) {
            super(itemView);
            txtData = itemView.findViewById(R.id.txt_data);
            txtTemp = itemView.findViewById(R.id.txt_temperatura);
            txtSensacao = itemView.findViewById(R.id.txt_sensacao);
            txtDescricao = itemView.findViewById(R.id.txt_descricao);
        }
    } // fim da classe interna ViewHolder e continua...
```

10. Consumindo API

• ClimaAdapter (para RecyclerView) 3/5:

```
@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int
viewType) {
    View view = LayoutInflater.from(parent.getContext()).
                inflate(R.layout.item_clima, parent, false);
    ViewHolder vh = new ViewHolder(view);
    return vh;
} // fim do onCreateViewHolder
// continua...
```

10. Consumindo API

• ClimaAdapter (para RecyclerView) 4/5:

```
@Override
public void onBindViewHolder(ViewHolder holder, int position) {
    String data = Utils.converterData(climas.get(position).getData());
    String temp = climas.get(position).getTemperatura()+"°C";
    String sensacao = climas.get(position).getSensacaoTermica()+"°C";
    String descricao = climas.get(position).getDescricao();
    holder.txtData.setText( data );
    holder.txtTemp.setText( temp );
    holder.txtSensacao.setText( sensacao );
    holder.txtDescricao.setText( descricao );
} // fim do onBindViewHolder
```

10. Consumindo API

• ClimaAdapter (para RecyclerView) 5/5:

```
    @Override
    public int getItemCount() {
        if(climas != null) return climas.size();
        else return 0;
    } // fim do getItemCount

} // fim da classe do Adapter
```

10. Consumindo API

• MainActivity 1/5:

```
public class MainActivity extends AppCompatActivity {
    private RecyclerView listaRV;
    private ClimaAdapter climaAdapter;
   @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // continua...
```

10. Consumindo API

• MainActivity 2/5:

```
listaRV = findViewById(R.id.rv_clima);

LinearLayoutManager lm = new LinearLayoutManager(this);

linearLayoutManager.setOrientation(LinearLayoutManager.VERTICAL);
listaRV.setLayoutManager(lm);
listaRV.setHasFixedSize(true);

// continua...
```

10. Consumindo API

• MainActivity 3/5:

```
if( Utils.isOnline(MainActivity.this) ) {
   String[] filtro = {"Belem,BR", "2bfe8d19224a72a9714c623429299b31",
"10"};
   new ClimaService(this, new AsyncTaskDelegate() {
      @Override
      public void processFinish(Object output) {
         List<Clima> climas = (List<Clima>) output;
         // continua...
```

10. Consumindo API

- MainActivity 4/5:

```
if(climas != null) {
    climaAdapter = new ClimaAdapter(climas);
    listaRV.setAdapter(climaAdapter);
} else {
    Toast.makeText(MainActivity.this, "Erro ao carregar a lista.",
Toast.LENGTH_SHORT).show();
} // fim do if-else
} // fim do método processFinish
}).execute(filtro); // fim do new ClimaService
```

10. Consumindo API

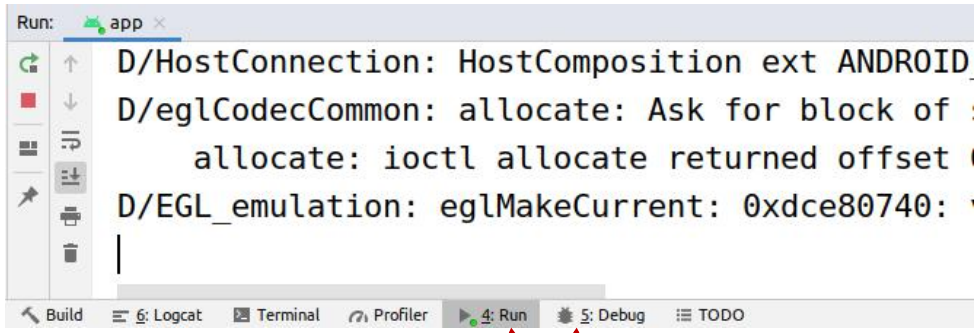
- MainActivity 5/5:

```
        } else {
            Toast.makeText(this, "Você está offline.",
Toast.LENGTH_SHORT).show();
        } // fim do if-else de checagem da internet
    } // fim do método onCreate
} // fim da classe MainActivity
```

10. Consumindo API

Hora dos testes!

Execute o aplicativo.



Erros são mostrados aqui

10. Consumindo API

Se tudo deu certo, o app vai rodar como planejado:

10. Consumindo API

• Melhorias:

1) Adicionar um indicativo de carregamento dos dados (widget ProgressBar).

2) Mostrar mensagem de offline na activity invés de usar Toast.

3) Criar um menu com um item para recarregar a lista e outro para mostrar informações sobre o app. Essas informações deverão aparecer em um AlertDialog.

4) Quando o usuário clicar em cima de uma previsão do tempo mostrada no RecyclerView, permitir que ele compartilhe a informação com outros apps.

5) Criar outros itens de menu para mostrar a previsão climática das seguintes cidades do Pará: Ananindeua, Castanhal, Salinópolis e outra a sua escolha.

10. Consumindo API

• Desafio #1:

Armazenar os dados oriundos da API no banco de dados local (SQLite) de modo que quando o usuário estiver offline, os últimos dados que estiverem no BD deverão ser mostrados. Além disso, economizaria chamadas desnecessárias à API, pacote de dados do usuário e bateria do smartphone.

10. Consumindo API

- Desafio #2:

Pegar as coordenadas do GPS do usuário e utilizá-las para pegar as previsões de tempo de acordo com a latitude e longitude que ele está posicionado.