Relatório trabalho

José Almeida- 16999/ Nuno Rodrigues-13282/ João Soares- 17628

Objetivo do trabalho

O nosso objetivo neste trabalho, era fazer uma lista que mostraria *Pokémon*, mostrando ao utilizador o número e o tipo de ataques de cada um. Para além disso, também seria possível ver uma imagem de cada um, e ao carregar em qualquer elemento, mostraria uma página da *Wiki*, com tudo que é possível saber sobre esse *Pokémon*.

Criação da classe "Pokémon"

Primeiro, decidimos que iriamos criar uma classe *Pokémon*, para adicionar que qualidades cada um dos elementos da lista iria ter. Começamos por definir os valores e o seu tipo, que depois seriam retirados de um ficheiro *.JSON*.

```
val name: String,
val imageUrl: String,
val url: String,
val type1: String,
val type2: String,
```

As variáveis que serão retiradas do ficheiro .JSON, serão guardados nestas variáveis. Agora falta definir o que vai ser procurado no ficheiro, e de que ficheiro. Para isso usamos o seguinte código:

```
fun getPokemonsFromFile(filename: String, context: Context): ArrayList<Pokemon> {
    val pokemonList = ArrayList<Pokemon>()

    try {
        // Load data
        val jsonString : String? = loadJsonFromAsset( filename: "pokeinfo.json", context)
        val json = JSONObject(jsonString)
        val pokemons : JSONArray = json.getJSONArray( name: "pokemons")

        // Get Pokemon objects from data
        (0 until pokemons.length()).mapTo(pokemonList) { it: Int
            Pokemon(pokemons.getJSONObject(it).getString( name: "name"),
                pokemons.getJSONObject(it).getString( name: "imageUrl"),
                pokemons.getJSONObject(it).getString( name: "url"),
                pokemons.getJSONObject(it).getString( name: "type1"),
                pokemons.getJSONObject(it).getString( name: "type2"),
                pokemons.getJSONObject(it).getString( name: "id"))
        }
    } catch (e: JSONException) {
        e.printStackTrace()
    }

    return pokemonList
}
```

```
private fun loadJsonFromAsset(filename: String, context: Context): String? {
    var json: String? = null

    try {
        val inputStream : InputStream = context.assets.open(filename)
        val size : Int = inputStream.available()
        val buffer = ByteArray(size)
        inputStream.read(buffer)
        inputStream.close()
        json = String(buffer, Charsets.UTF_8)
    } catch (ex: java.io.IOException) {
        ex.printStackTrace()
        return null
    }

    return json
}
```

Criação da List View e Adapter.

De seguida, fomos ao ficheiro *activity_main.xml* e adicionamos uma List View. A essa view demos o id de “pokemon_list_view”

```
<ListView
    android:id="@+id/pokemon_list_view"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

De seguida inicializamos uma instância da List View na MainActivity

```
private lateinit var listView: ListView
```

Dentro do método de onCreate adicionamos o seguinte código:

```
listView = findViewById<ListView>(R.id.pokemon_list_view)

val pokemonList : ArrayList<Pokemon> = Pokemon.getPokemonsFromFile( filename: "pokeinfo.json", context: this)

val adapter = PokemonAdapter( context: this, pokemonList)
listView.adapter = adapter
```

Este pedaço de código está a criar um array de strings que será apresentado no *ListView*. Como é possível ver colocamos ali o *PokemonAdapter*, e foi o que adicionamos de seguida.

Criamos uma nova classe chamada de *PokemonAdapter*, que servirá para buscar os itens que irão ser mostrados no *ListView.*

```
listView = findViewById<ListView>(R.id.pokemon_list_view)

val pokemonList : ArrayList<Pokemon> = Pokemon.getPokemonsFromFile( filename: "pokeinfo.json", context: this)

val adapter = PokemonAdapter( context: this, pokemonList)
listView.adapter = adapter
```

```
override fun getCount(): Int {
    return dataSource.size
}


override fun getItem(position: Int): Any {
    return dataSource[position]
}


override fun getItemId(position: Int): Long {
    return position.toLong()
}
```

Após esta adição de um adapter, é necessário criar um layout, para mostrar texto assim como imagens. Criamos num novo ficheiro .xml e demos o nome de *list_item_pokemon.* Dentro deste layout, adicionamos 4 *textViews* e 1 *imageView*.

```
<ImageView
    android:id="@+id/pokemon_list_image"
    android:layout_width="90dp"
    android:layout_height="90dp"
    android:layout_alignParentStart="true"
    android:layout_centerVertical="true"
    android:layout_marginBottom="6dp"
    android:layout_marginStart="4dp"
    android:layout_marginTop="6dp"
    android:contentDescription="thumbnail"
    android:scaleType="centerInside"
    tools:src="@mipmap/ic_launcher"
    android:layout_alignParentLeft="true"
    android:layout_marginLeft="4dp" />

<TextView
    android:id="@+id/pokemon_list_id"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_centerVertical="true"
    android:layout_marginEnd="2dp"
    android:layout_marginStart="4dp"
    android:maxLines="1"
    android:paddingEnd="4dp"
    android:textColor="#000000"
    android:textSize="12sp"
    tools:text="Detail"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="2dp"
    android:layout_alignParentRight="true"
    android:paddingRight="4dp" />
```

```
<RelativeLayout
    android:id="@+id/recipe_list_text_layout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerVertical="true"
    android:layout_toEndOf="@id/pokemon_list_image"
    android:layout_toStartOf="@id/pokemon_list_id"
    android:layout_toRightOf="@id/pokemon_list_image"
    android:layout_toLeftOf="@id/pokemon_list_id">

    <TextView
        android:id="@+id/pokemon_list_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:textSize="24sp"
        android:textStyle="bold"
        tools:text="Title" />

    <TextView
        android:id="@+id/pokemon_list_type"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/pokemon_list_name"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="2dp"
        android:ellipsize="end"
        android:maxLines="3"
        android:textSize="16sp"
        tools:text="Subtitle" />

    <TextView
        android:id="@+id/pokemon_list_type2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignTop="@+id/pokemon_list_type"
        android:layout_marginStart="3dp"
        android:layout_marginLeft="3dp"
        android:layout_marginTop="0dp"
        android:layout_toEndOf="@+id/pokemon_list_type"
        android:layout_toRightOf="@+id/pokemon_list_type"
        android:text="Subtitle2"
        android:textSize="16sp" />

</RelativeLayout>
```

Voltando ao Adapter, mais propriamente ao *getView(),* vamos dizer que informação vai para cada *textView* e carregar a imagem através do *Picasso*, que vai buscar *Url* da imagem que se pretende mostrar e transforma-a de maneira a poder ser visualizada no *imageView.*

```
override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
    // Get view for row item
    val rowView : View? = inflater.inflate(R.layout.list_item_pokemon, parent, attachToRoot: false)

    val nameTextView : TextView = rowView.findViewById(R.id.pokemon_list_name) as TextView

    val typeTextView : TextView = rowView.findViewById(R.id.pokemon_list_type) as TextView

    val typeTextView2 : TextView = rowView.findViewById(R.id.pokemon_list_type2) as TextView

    val idTextView : TextView =rowView.findViewById(R.id.pokemon_list_id) as TextView

    val imageImageView : ImageView = rowView.findViewById(R.id.pokemon_list_image) as ImageView

    val pokemon : Pokemon = getItem(position) as Pokemon

    nameTextView.text = pokemon.name
    typeTextView.text = pokemon.type1
    typeTextView2.text = pokemon.type2
    idTextView.text = pokemon.id

    Picasso.with(context).load(pokemon.imageUrl).placeholder(R.mipmap.ic_launcher).into(imageImageView)

    typeTextView.setTextColor( ContextCompat.getColor(context, id: LABEL_COLORS[pokemon.type1] ?: R.color.colorPrimary))
    typeTextView2.setTextColor( ContextCompat.getColor(context, id: LABEL_COLORS[pokemon.type2] ?: R.color.colorPrimary))

    return rowView
}
```

WebView detail activity

A próxima fase do trabalho queríamos que o utilizador pudesse carregar no *Pokémon* à sua escolha, e ao carregar a aplicação iria o levar ao site correspondente a esse *Pokémon*. Para isso precisamos de adicionar um *WebView.*

Começamos por adicionar um novo ficheiro .xml ao layout chamado de *activity_pokemon_detail* e dentro adicionamos um *WebView.*

```
<WebView
    android:id="@+id/detail_web_view"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Depois criamos uma nova classe, chamada de *PokemonDetailActivity*. Dentro desta classe adicionamos uma referência ao *WebView*:

```
private lateinit var webView: WebView
```

E de seguida a declaração:

```
companion object {
    const val EXTRA_TITLE = "title"
    const val EXTRA_URL = "url"

    fun newIntent(context: Context, pokemon: Pokemon): Intent {
        val detailIntent = Intent(context, PokeDetailActivity::class.java)

        detailIntent.putExtra(EXTRA_TITLE, pokemon.name)
        detailIntent.putExtra(EXTRA_URL, pokemon.url)

        return detailIntent
    }
}
```

Este pedaço de código cria o *Intent* de começar o *DetailActivity* assim como definir o nome e o Url. Voltando à *MainActivity* adicionamos o seguinte código ao *onCreate* ir buscar o *Pokémon* que se está a carregar.

```
val context : MainActivity = this
listView.setOnItemClickListener { _, _, position, _ ->
    // 1
    val selectedPokemon : Pokemon = pokemonList[position]

    // 2
    val detailIntent : Intent = PokeDetailActivity.newIntent(context, selectedPokemon)

    // 3
    startActivity(detailIntent)
}
```

Voltando de novo ao *PokemonDetailActivity*, falta adicionar ao fim do *onCreate* o seguinte código:

```
val title : String?  = intent.extras?.getString(EXTRA_TITLE)
val url : String?  = intent.extras?.getString(EXTRA_URL)


setTitle(title)


webView = findViewById(R.id.detail_web_view)


webView.loadUrl(url)
```

Neste pedaço de código, vai se buscar a informação dos Pokémons ao intente da MainActivity ao usar os extras. O setTitle, como o nome indica está a definir o nome que aparece em cima, para o nome do Pokémon que estará a ser visualizado. Inicializa-se também o webView e dá se load ao Url do Pokémon selecionado.

Adição de cores correspondentes aos tipos

Para finalizar, decidimos que seria boa ideia que na *listView* o tipo de *Pokémon* seria de cor diferente conforme a sua cor dentro dos jogos. Fomos ao *colors.xml,* e escrevemos os valores da cor e demos o nome correspondente.

```
<resources>
    <color name="colorPrimary">#008577</color>
    <color name="colorPrimaryDark">#00574B</color>
    <color name="colorAccent">#D81B60</color>
    <color name="colorNormal">#A8A77A</color>
    <color name="colorFire">#EE8130</color>
    <color name="colorWater">#6390F0</color>
    <color name="colorElectric">#F7D02C</color>
    <color name="colorGrass">#7AC74C</color>
    <color name="colorIce">#96D9D6</color>
    <color name="colorFighting">#C22E28</color>
    <color name="colorPoison">#A33EA1</color>
    <color name="colorGround">#E2BF65</color>
    <color name="colorFlying">#A98FF3</color>
    <color name="colorPsychic">#F95587</color>
    <color name="colorBug">#A6B91A</color>
    <color name="colorRock">#B6A136</color>
    <color name="colorGhost">#735797</color>
    <color name="colorDragon">#6F35FC</color>
    <color name="colorDark">#705746</color>
    <color name="colorSteel">#B7B7CE</color>
    <color name="colorFairy">#D685AD</color>
</resources>
```

Abrimos outra vez o *PokemonAdapter* e adicionamos o seguinte código:

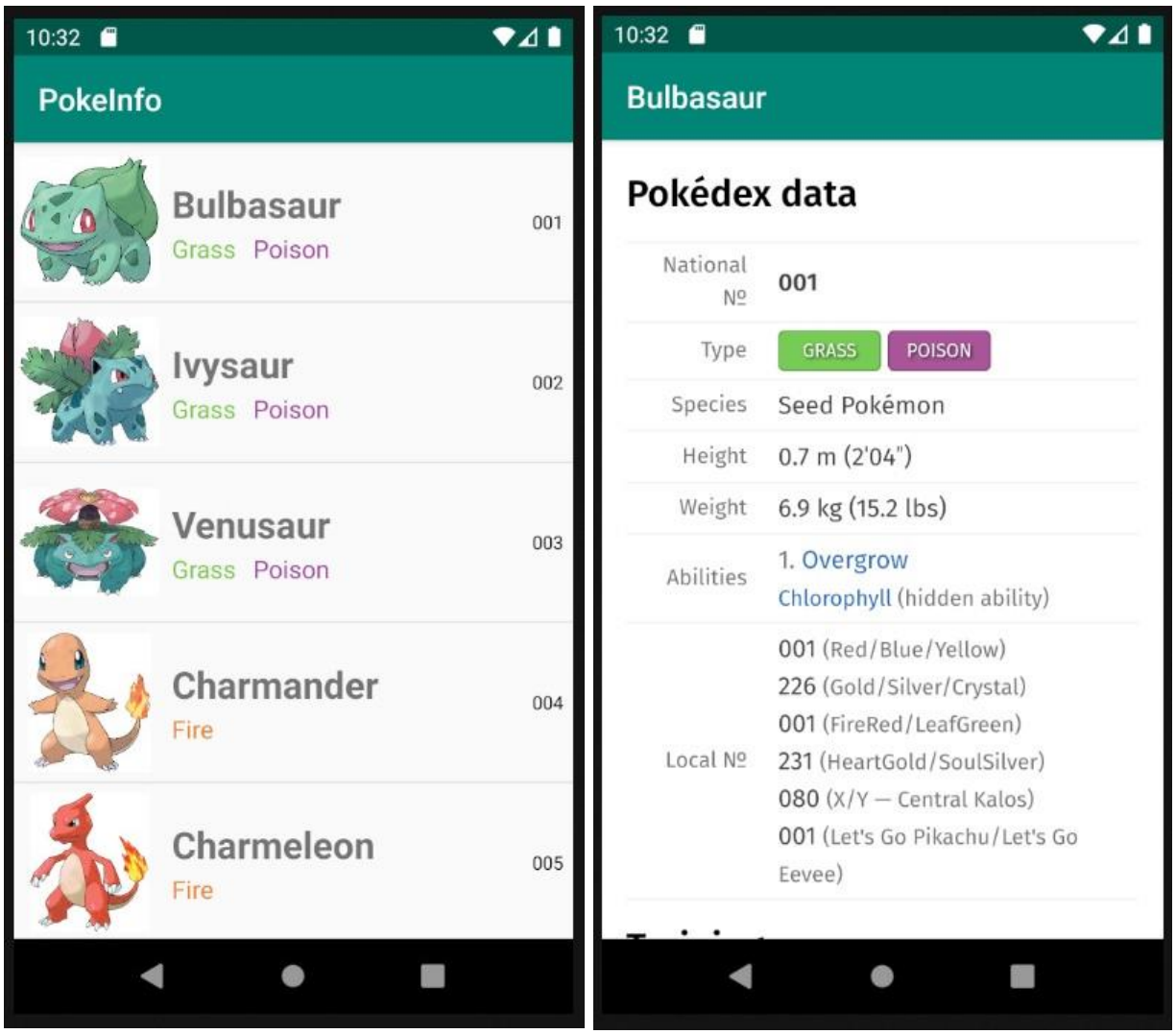
```
18      companion object{
19          private val LABEL_COLORS : HashMap<String, Int>  = hashMapOf(
20              "Normal" to R.color.colorNormal,
21              "Fire" to R.color.colorFire,
22              "Water" to R.color.colorWater,
23              "Electric" to R.color.colorElectric,
24              "Grass" to R.color.colorGrass,
25              "Ice" to R.color.colorIce,
26              "Fighting" to R.color.colorFighting,
27              "Poison" to R.color.colorPoison,
28              "Ground" to R.color.colorGround,
29              "Flying" to R.color.colorFlying,
30              "Psychic" to R.color.colorPsychic,
31              "Bug" to R.color.colorBug,
32              "Rock" to R.color.colorRock,
33              "Ghost" to R.color.colorGhost,
34              "Dragon" to R.color.colorDragon,
35              "Dark" to R.color.colorDark,
36              "Steel" to R.color.colorSteel,
37              "Fairy" to R.color.colorFairy
38
39          )
40      }
```

Assim estamos a definir que se algum texto corresponder a este, irá utilizar a cor correspondente a esse texto. No *getView()* adicionamos o próximo código, para definir que *TextView* irá procurar o texto correspondente e atribuir a sua cor.

```
typeTextView.setTextColor( ContextCompat.getColor(context,  id: LABEL_COLORS[pokemon.type1] ?: R.color.colorPrimary))
typeTextView2.setTextColor( ContextCompat.getColor(context,  id: LABEL_COLORS[pokemon.type2] ?: R.color.colorPrimary))
```

Aspeto final da aplicação

Após o código ter sido implementado, o aspeto da aplicação é o seguinte:



Conclusão

Para concluir o trabalho, apesar de todos os problemas que encontramos na criação da aplicação, gostamos de desenvolver a aplicação e gostamos do resultado final da aplicação. Gostaríamos de ter adicionado uma *search bar* para ser mais fácil a pesquisa, mas visto que estávamos a utilizar uma *listView*, teríamos que reescrever bastante código, então esquecemos essa ideia. Tivemos também que escrever o ficheiro completo de .JSON, mas valeu a pena para o resultado final.