



Reskilling 4Employment Software Developer

Acesso móvel a sistemas de informação

Bruno Santos

bruno.santos.mcv@msft.cesae.pt

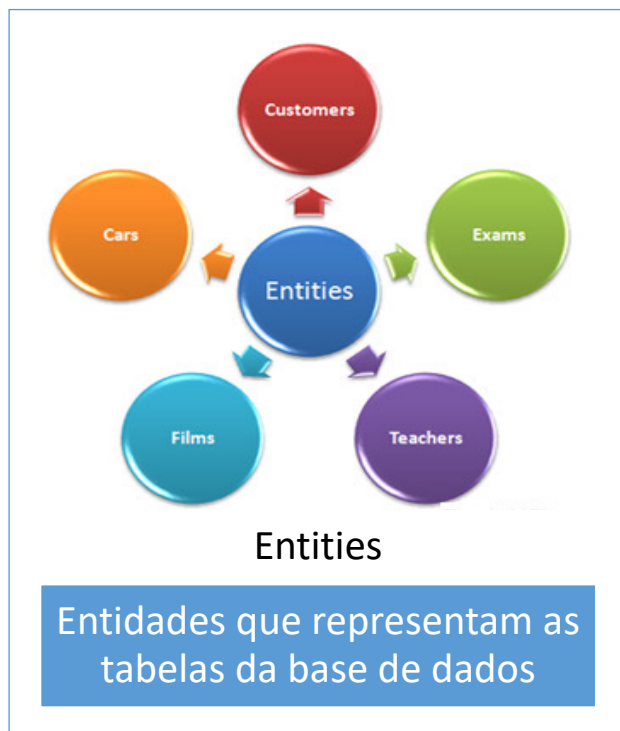
Tópicos

- Room

Android Room

- A biblioteca de persistência Room oferece uma camada de abstração sobre o SQLite para permitir um acesso mais robusto ao banco de dados, aproveitando toda a capacidade do SQLite.

Android Room



Android Room

- No ficheiro build.gradle vamos adicionar, no bloco plugin:
 - id 'kotlin-kapt'
- No mesmo ficheiro na dependencies adicionar:
 - implementation 'androidx.room:room-runtime:2.4.3'
 - kapt 'androidx.room:room-compiler:2.4.3'
- Sincronizar o ficheiro gradle para incluir as dependências do Room

Android Room

```
1  plugins {  
2      id 'com.android.application'  
3      id 'org.jetbrains.kotlin.android'  
4      id 'kotlin-kapt'
```

```
49      implementation 'androidx.room:room-runtime:2.4.3'  
50      kapt 'androidx.room:room-compiler:2.4.3'
```

Android Room

- Como exemplo vamos criar uma aplicação para gerir utilizadores de uma aplicação.
- Primeiro passo vamos criar a classe de Modelo: UserModel

Android Room

```
1 package com.example.a16_room
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.PrimaryKey
6
7 @Entity(tableName = "User")
8 class UserModel {
9
10     @PrimaryKey(autoGenerate = true)
11     @ColumnInfo(name = "id")
12     var id: Int = 0
13
14     @ColumnInfo(name = "username")
15     var username: String = ""
16
17     @ColumnInfo(name = "password")
18     var password: String = ""
19 }
```

- Foi definida a anotação `@Entity` para informação de que `UserModel` será a classe que vai mapear a tabela (tableName) `User`.

Android Room

```
1 package com.example.a16_room
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.PrimaryKey
6
7 @Entity(tableName = "User")
8 class UserModel {
9
10     @PrimaryKey(autoGenerate = true)
11     @ColumnInfo(name = "id")
12     var id: Int = 0
13
14     @ColumnInfo(name = "username")
15     var username: String = ""
16
17     @ColumnInfo(name = "password")
18     var password: String = ""
19 }
```

- Em cada campo é colocada a anotação `@ColumnInfo` para identificar a coluna na tabela da `@Entity`
- O parâmetro `name` é opcional, sendo que se não for colocado é reaproveitado o nome da variável.

Android Room

```
1 package com.example.a16_room
2
3 import androidx.room.ColumnInfo
4 import androidx.room.Entity
5 import androidx.room.PrimaryKey
6
7 @Entity(tableName = "User")
8 class UserModel {
9
10     @PrimaryKey(autoGenerate = true)
11     @ColumnInfo(name = "id")
12     var id: Int = 0
13
14     @ColumnInfo(name = "username")
15     var username: String = ""
16
17     @ColumnInfo(name = "password")
18     var password: String = ""
19 }
```

- No campo id foi colocada a identificação da `@PrimaryKey`, bem como o parâmetro `autoGenerate = true` para fazer o incremento automático

Android Room

- Agora vamos criar a classe responsável que manipulação direta da base de dados: `UserDatabase`
- Esta classe será uma classe abstrata que estende de `RoomDatabase` e vai ter a anotação de `@Database`
- A anotação de `@Database` implica a passagem dos parâmetros:
 - `Entities` – indicação das classes (modelos) a implementar na base de dados
 - `Version` – versão da base de dados

```
@Database(entities = [UserModel::class], version = 1)
abstract class UserDatabase : RoomDatabase() {
}
```

Android Room

- Dentro da classe vamos criar um companion object e aplicar o padrão Singleton para aceder à base de dados
- O padrão Singleton tem como definição garantir que uma classe tenha apenas uma instância de si mesma e que forneça um ponto global de acesso a ela. Ou seja, uma classe gere a sua própria instância, além de evitar que qualquer outra classe crie uma nova instância desta.

Android Room

```
companion object {  
    private lateinit var INSTANCE: UserDatabase  
  
    fun getDatabase(context: Context): UserDatabase {  
        if (!::INSTANCE.isInitialized) {  
            synchronized(UserDatabase::class.java) {  
                INSTANCE = Room.databaseBuilder(context, UserDatabase::class.java, name: "userDB")  
                    .allowMainThreadQueries()  
                    .build()  
            }  
        }  
        return INSTANCE  
    }  
}
```

Android Room

- Na função `getDatabase` vamos validar se a instância desta classe já foi ou não instanciada, e, em caso negativo, instancia. Em qualquer caso devolve a ligação à base de dados instanciada.
- O parâmetro `synchronized` permite trabalhar a chamada à base de dados sem acessos múltiplos, de forma a evitar a criação de múltiplas instanciações iniciais simultâneas.
- Adicionamos agora a possibilidade de acrescentar uma `Migration` à base de dados (semelhante ao `onUpdate`), passando uma implementação da `Migration` com a versão antiga e nova da base de dados

Android Room

```
fun getDatabase(context: Context): UserDatabase {  
    if (!::INSTANCE.isInitialized) {  
        synchronized(UserDatabase::class.java) {  
            INSTANCE = Room.databaseBuilder(context, UserDatabase::class.java, name: "userDB")  
                .addMigrations(MIGRATION_1_2)  
                .allowMainThreadQueries()  
                .build()  
        }  
    }  
    return INSTANCE  
}
```

```
private val MIGRATION_1_2: Migration = object : Migration(1, 2) {  
    override fun migrate(database: SupportSQLiteDatabase) {  
        //IMPLEMENTAR O NECESSÁRIO  
    }  
}
```

Android Room

- Implementadas as camadas de Entities e Database, necessitamos agora de criar a DAO para fazer as Entities e a Database se comunicarem, assim vamos começar por criar uma Interface com os métodos a implementar.
- Esta interface terá a anotação @Dao

Android Room

```
@Dao
interface UserDAO {

    @Insert
    fun insert(user: UserModel): Long

    @Update
    fun update(user: UserModel): Int

    @Delete
    fun delete(user: UserModel): Int

    @Query("SELECT * FROM User WHERE id = :id")
    fun get(id: Int): UserModel

    @Query("SELECT * FROM User")
    fun getAll(): List<UserModel>
}
```

- As funções insert, update e delete têm as anotações @Insert, @Update, @Delete, que dão a informação de que são os métodos específicos para fazerem as três opções de alteração de dados

Android Room

```
@Dao
interface UserDAO {

    @Insert
    fun insert(user: UserModel): Long

    @Update
    fun update(user: UserModel): Int

    @Delete
    fun delete(user: UserModel): Int

    @Query("SELECT * FROM User WHERE id = :id")
    fun get(id: Int): UserModel

    @Query("SELECT * FROM User")
    fun getAll(): List<UserModel>
}
```

- As funções `get` e `getAll` têm a anotação `@Query` e a informação de qual a query a ser executada quando chamado o método. Os parâmetros são passados na query com `:` antes do mesmo por segurança

Android Room

- Na classe UserDatabase vamos criar uma função abstrata para ligar a UserDAO à base de dados

```
abstract class UserDatabase : RoomDatabase() {  
    abstract fun userDAO(): UserDAO  
  
    companion object {  
        private lateinit var INSTANCE: UserDatabase  
    }  
}
```

Android Room

- Aplicando as boas práticas de separação de código, vamos criar a classe UserRepository que será a camada de da nossa aplicação responsável pelo acesso aos dados, nesta vamos implementar os métodos da interface criada anteriormente

Android Room

```
class UserRepository(context: Context) {  
  
    private val userDatabase = UserDatabase.getDatabase(context).userDAO()  
  
    fun insert(user: UserModel): Long {  
        return userDatabase.insert(user)  
    }  
  
    fun update(user: UserModel): Int {  
        return userDatabase.update(user)  
    }  
  
    fun delete(user: UserModel): Int {  
        return userDatabase.delete(user)  
    }  
  
    fun get(id: Int): UserModel {  
        return userDatabase.get(id)  
    }  
  
    fun getAll(): List<UserModel> {  
        return userDatabase.getAll()  
    }  
}
```

Android Room

- De seguida criamos uma interface para testar todos os métodos

Android Room

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res-a
3     xmlns:app="http://schemas.android.com/apk/res-a
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     android:orientation="vertical"
8     android:padding="8dp"
9     tools:context=".MainActivity">
10     <TextView
11         android:id="@+id/text_id"
12         android:layout_width="match_parent"
13         android:layout_height="wrap_content"
14         android:gravity="center"
15         android:text="ID:" />
16     <EditText
17         android:id="@+id/edit_username"
18         android:layout_width="match_parent"
19         android:layout_height="wrap_content"
20         android:gravity="center"
21         android:hint="Username" />
22     <EditText
23         android:id="@+id/edit_password"
24         android:layout_width="match_parent"
25         android:layout_height="wrap_content"
26         android:gravity="center"
27         android:hint="Password" />
28
```

```
28
29 <Button
30     android:id="@+id/button_insert"
31     android:layout_width="match_parent"
32     android:layout_height="wrap_content"
33     android:text="Insert" />
34
35 <Button
36     android:id="@+id/button_edit"
37     android:layout_width="match_parent"
38     android:layout_height="wrap_content"
39     android:text="Edit" />
40
41 <Button
42     android:id="@+id/button_delete"
43     android:layout_width="match_parent"
44     android:layout_height="wrap_content"
45     android:text="Delete" />
46
47 <androidx.recyclerview.widget.RecyclerView
48     android:id="@+id/recycler_users"
49     android:layout_width="match_parent"
50     android:layout_height="wrap_content" />
51
52 </LinearLayout>
```

Android Room

ID:

Username

Password

Insert

Edit

Delete

Item 0

Item 1

Item 2

Item 3

Item 4

Item 5

Item 6

Item 7

Item 8

Item 9

Android Room

MainActivity.kt

```
8 class MainActivity : AppCompatActivity() {
9
10     private lateinit var binding: ActivityMainBinding
11     private lateinit var viewModel: MainViewModel
12
13     override fun onCreate(savedInstanceState: Bundle?) {
14         super.onCreate(savedInstanceState)
15         binding = ActivityMainBinding.inflate(layoutInflater)
16         setContentView(binding.root)
17
18         viewModel = ViewModelProvider(owner: this).get(MainViewModel::class.java)
19
20         observe()
21         loadData()
22     }
23
24     private fun observe() {
25         TODO(reason: "Not yet implemented")
26     }
27
28     private fun loadData() {
29         TODO(reason: "Not yet implemented")
30     }
31 }
```

MainViewModel.kt

```
3 import androidx.lifecycle.ViewModel
4
5 class MainViewModel : ViewModel() {
6 }
```

Android Room

- Uma vez que estamos a utilizar uma RecyclerView, necessitamos criar:
 - Arquivo de layout da linha da RecyclerView (row_user.xml)
 - Adapter (UserAdapter.kt)
 - ViewHolder (UserViewHolder.kt)
 - Interface OnUserListener (OnUserListener.kt)

Android Room (row_user.xml)

```
1      <?xml version="1.0" encoding="utf-8"?>
2      <LinearLayout xmlns:android="http://schemas.android.com
3          android:layout_width="match_parent"
4          android:layout_height="wrap_content"
5          android:orientation="horizontal"
6          android:padding="8dp">
7
8          <TextView
9              android:id="@+id/text_username"
10             android:layout_width="match_parent"
11             android:layout_height="wrap_content"
12             android:gravity="center"
13             android:text="Username"
14             android:textSize="20sp" />
15
16      </LinearLayout>
```

Android Room (UserViewHolder.kt)

```
class UserViewHolder(private val bind: RowUserBinding, private val listener: OnUserClick) :  
    RecyclerView.ViewHolder(bind.root) {  
  
    fun bind(user: UserModel) {  
        bind.textUsername.text = user.username  
  
        bind.textUsername.setOnClickListener { it: View! -> listener.OnClick(user.id)  
        }  
    }  
}
```

Android Room (UserAdapter.kt)

```
class UserAdapter : RecyclerView.Adapter<UserViewHolder>() {  
  
    private var userList: List<UserModel> = listOf()  
    private lateinit var listener: OnUserListeneer  
  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): UserViewHolder {  
        val item = RowUserBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)  
        return UserViewHolder(item, listener)  
    }  
  
    override fun onBindViewHolder(holder: UserViewHolder, position: Int) {  
        holder.bind(userList[position])  
    }  
  
    override fun getItemCount(): Int {  
        return userList.size  
    }  
  
    fun updateUsers(list: List<UserModel>) {  
        userList = list  
        notifyDataSetChanged()  
    }  
  
    fun attachListener(userListener: OnUserListeneer) {  
        listener = userListener  
    }  
}
```

Android Room (OnUserListener.kt)

```
interface OnUserListener {  
    fun OnClick(id: Int)  
}
```

Android Room

- E agora finalmente implementar o restante da MainActivity e MainViewModel

Android Room (MainActivity.kt)



Centro para o Desenvolvimento
de Competências Digitais

```
11 class MainActivity : AppCompatActivity() {
12
13     private lateinit var binding: ActivityMainBinding
14     private lateinit var viewModel: MainViewModel
15     private val adapter = UserAdapter()
16     private var id = 0
17
18     override fun onCreate(savedInstanceState: Bundle?) {
19         super.onCreate(savedInstanceState)
20         binding = ActivityMainBinding.inflate(layoutInflater)
21         setContentView(binding.root)
22
23         viewModel = ViewModelProvider(owner: this).get(MainViewModel::class.java)
24
25         binding.recyclerUsers.layoutManager = LinearLayoutManager(applicationContext)
26         binding.recyclerUsers.adapter = adapter
27
28         val listener = object : OnUserListener {
29             override fun onClick(id: Int) {
30                 Toast.makeText(applicationContext, id.toString(), Toast.LENGTH_SHORT).show()
31                 viewModel.get(id)
32             }
33         }
34         adapter.attachListener(listener)
35
36         binding.buttonInsert.setOnClickListener { it: View!
37             val username = binding.editUsername.text.toString()
38             val password = binding.editPassword.text.toString()
39             viewModel.insert(username, password)
40         }
```

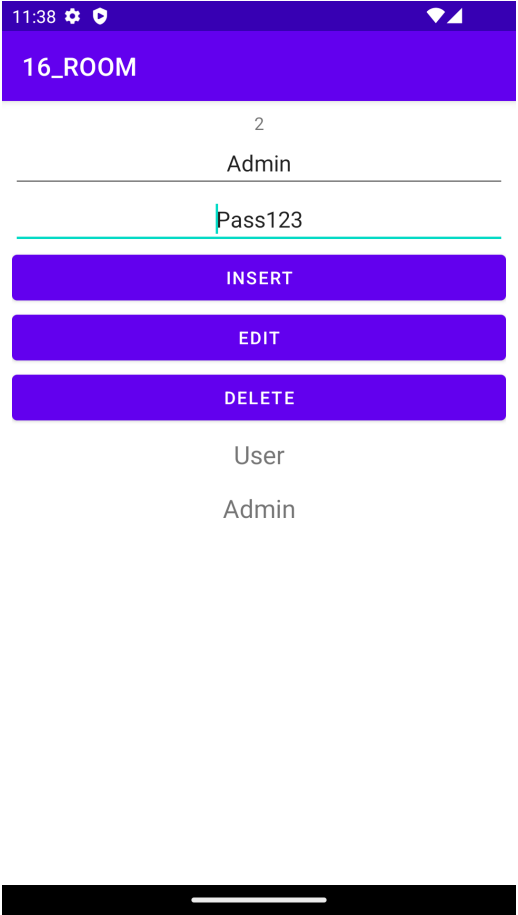
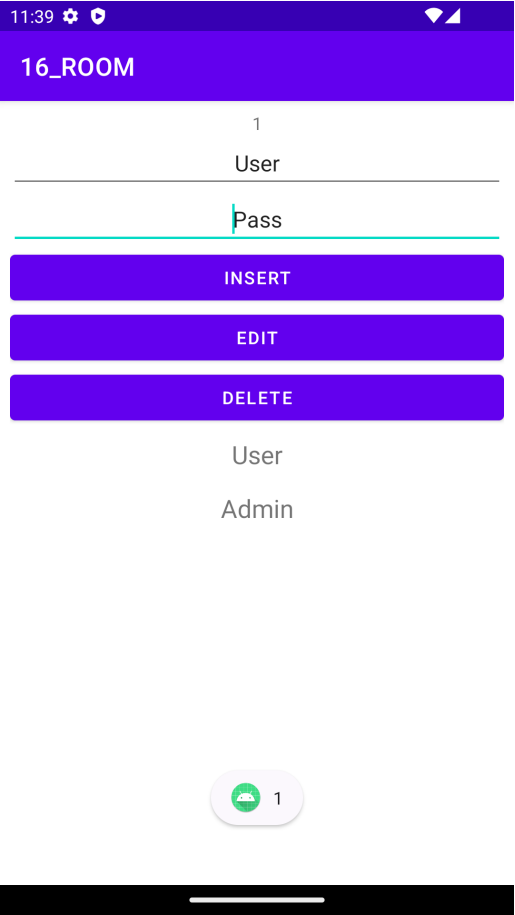
```
41         binding.buttonEdit.setOnClickListener { it: View!
42             val username = binding.editUsername.text.toString()
43             val password = binding.editPassword.text.toString()
44             viewModel.update(id, username, password)
45         }
46         binding.buttonDelete.setOnClickListener { it: View!
47             viewModel.delete(id)
48         }
49
50         observe()
51         viewModel.getAll()
52     }
53
54     private fun observe() {
55         viewModel.users.observe(owner: this) { it: List<UserModel>!
56             adapter.updateUsers(it)
57         }
58         viewModel.user.observe(owner: this) { it: UserModel!
59             id = it.id
60             binding.textId.setText(id.toString())
61             binding.editUsername.setText(it.username)
62             binding.editPassword.setText(it.password)
63         }
64         viewModel.newChange.observe(owner: this) { it: Long!
65             viewModel.getAll()
66         }
67     }
68 }
```


Android Room (MainViewModel.kt)

```
8 class MainViewModel(application: Application) : AndroidViewModel(application) {
9
10     private val repository = UserRepository(application.applicationContext)
11
12     private val listUsers = MutableLiveData<List<UserModel>>()
13     val users: LiveData<List<UserModel>> = listUsers
14
15     private val userModel = MutableLiveData<UserModel>()
16     var user: LiveData<UserModel> = userModel
17
18     private var changes = MutableLiveData<Long>()
19     var newChange: LiveData<Long> = changes
20
21     fun getAll() {
22         listUsers.value = repository.getAll()
23     }
24
25     fun get(id: Int) {
26         userModel.value = repository.get(id)
27     }
28 }
```

```
30 fun insert(username: String, password: String) {
31     val model = UserModel().apply { this: UserModel
32         this.username = username
33         this.password = password
34     }
35     changes.value = repository.insert(model)
36 }
37
38 fun update(id: Int, username: String, password: String) {
39     val model = UserModel().apply { this: UserModel
40         this.id = id
41         this.username = username
42         this.password = password
43     }
44     changes.value = repository.update(model).toLong()
45 }
46
47 fun delete(id: Int) {
48     val model = UserModel().apply { this: UserModel
49         this.id = id
50     }
51     changes.value = repository.delete(model).toLong()
52 }
```

Android Room



Android Room