INSTITUTO DO EMPREGO
E FORMAÇÃO PROFISSIONAL

AEP

cesae digital

Centro para o Desenvolvimento
de Competências Digitais

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

Entities

Entidades que representam as tabelas da base de dados

DAO

Design Pattern

DAO (Data Access Objects)

Métodos de acesso à base de dados

Database

Ponto de acesso à base de dados

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

```
plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
    id 'kotlin-kapt'
}
```

```
    implementation 'androidx.room:room-runtime:2.4.3'
    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

```kotlin
package com.example.a16_room

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "User")
class UserModel {

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    var id: Int = 0

    @ColumnInfo(name = "username")
    var username: String = ""

    @ColumnInfo(name = "password")
    var password: String = ""
}
```

- 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

```kotlin
package com.example.a16_room

import androidx.room.ColumnInfo
import androidx.room.Entity
import androidx.room.PrimaryKey

@Entity(tableName = "User")
class UserModel {

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "id")
    var id: Int = 0

    @ColumnInfo(name = "username")
    var username: String = ""

    @ColumnInfo(name = "password")
    var password: String = ""
}
```

- No campo id foi colocada a identificação da @PrimaryKey, bem como o parâmetro autoGenerate a 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 syncronized 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

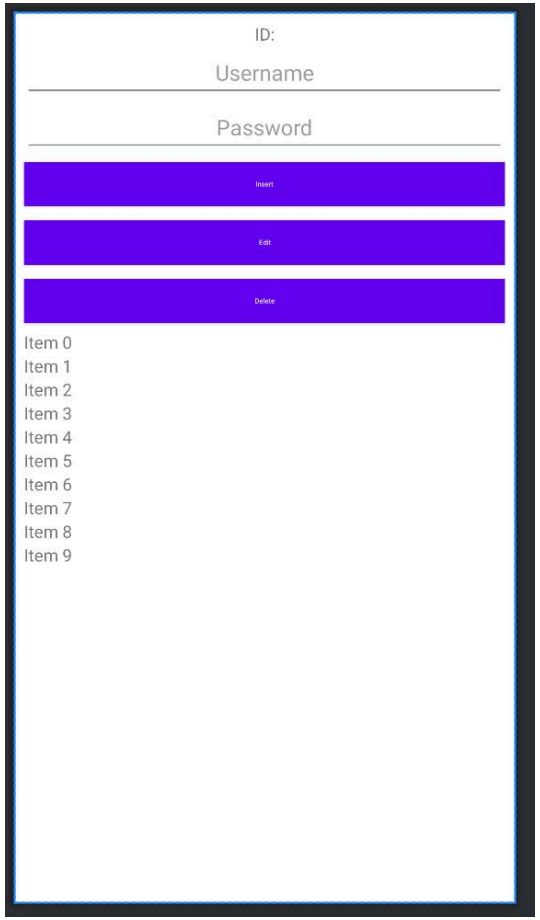
- De seguida criamos uma interface para testar todos os métodos

Android Room

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android
    xmlns:app="http://schemas.android.com/apk/res-a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="8dp"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/text_id"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="ID:" />
    <EditText
        android:id="@+id/edit_username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:hint="Username" />
    <EditText
        android:id="@+id/edit_password"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:hint="Password" />
```

```xml
    <Button
        android:id="@+id/button_insert"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Insert" />

    <Button
        android:id="@+id/button_edit"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Edit" />

    <Button
        android:id="@+id/button_delete"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Delete" />

    <androidx.recyclerview.widget.RecyclerView
        android:id="@+id/recycler_users"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</LinearLayout>
```

Android Room



Android Room

MainActivity.kt

```kotlin
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var viewModel: MainViewModel

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider( owner: this).get(MainViewModel::class.java)

        observe()
        loadData()
    }

    private fun observe() {
        TODO( reason: "Not yet implemented")
    }

    private fun loadData() {
        TODO( reason: "Not yet implemented")
    }
}
```

MainViewModel.kt

```kotlin
import androidx.lifecycle.ViewModel

class MainViewModel : ViewModel() {
}
```

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)

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:padding="8dp">

    <TextView
        android:id="@+id/text_username"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Username"
        android:textSize="20sp" />

</LinearLayout>
```

Android Room (UserViewHolder.kt)

```kotlin
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)

```kotlin
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)

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

Android Room

- E agora finalmente implementar o restante da MainActivity e MainViewModel

Android Room (MainActivity.kt)

```kotlin
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding
    private lateinit var viewModel: MainViewModel
    private val adapter = UserAdapter()
    private var id = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        viewModel = ViewModelProvider( owner: this).get(MainViewModel::class.java)

        binding.recyclerUsers.layoutManager = LinearLayoutManager(applicationContext)
        binding.recyclerUsers.adapter = adapter

        val listener = object : OnUserListener {
            override fun OnClick(id: Int) {
                Toast.makeText(applicationContext, id.toString(), Toast.LENGTH_SHORT).show()
                viewModel.get(id)
            }
        }
        adapter.attachListener(listener)

        binding.buttonInsert.setOnClickListener {  it: View!
            val username = binding.editUsername.text.toString()
            val password = binding.editPassword.text.toString()
            viewModel.insert(username, password)
        }
        binding.buttonEdit.setOnClickListener {  it: View!
            val username = binding.editUsername.text.toString()
            val password = binding.editUsername.text.toString()
            viewModel.update(id, username, password)
        }
        binding.buttonDelete.setOnClickListener {  it: View!
            viewModel.delete(id)
        }

        observe()
        viewModel.getAll()
    }

    private fun observe() {
        viewModel.users.observe( owner: this) {  it: List<UserModel>!
            adapter.updateUsers(it)
        }
        viewModel.user.observe( owner: this) {  it: UserModel!
            id = it.id
            binding.textId.setText(id.toString())
            binding.editUsername.setText(it.username)
            binding.editPassword.setText(it.password)
        }
        viewModel.newChange.observe( owner: this) {  it: Long!
            viewModel.getAll()
        }
    }
}
```

Android Room (MainViewModel.kt)

```kotlin
class MainViewModel(application: Application) : AndroidViewModel(application) {

    private val repository = UserRepository(application.applicationContext)

    private val listUsers = MutableLiveData<List<UserModel>>()
    val users: LiveData<List<UserModel>> = listUsers

    private val userModel = MutableLiveData<UserModel>()
    var user: LiveData<UserModel> = userModel

    private var changes = MutableLiveData<Long>()
    var newChange: LiveData<Long> = changes

    fun getAll() {
        listUsers.value = repository.getAll()
    }

    fun get(id: Int) {
        userModel.value = repository.get(id)

    }
```

```kotlin
    fun insert(username: String, password: String) {
        val model = UserModel().apply { this: UserModel
            this.username = username
            this.password = password
        }
        changes.value = repository.insert(model)
    }

    fun update(id: Int, username: String, password: String) {
        val model = UserModel().apply { this: UserModel
            this.id = id
            this.username = username
            this.password = password
        }
        changes.value = repository.update(model).toLong()
    }

    fun delete(id: Int) {
        val model = UserModel().apply { this: UserModel
            this.id = id
        }
        changes.value = repository.delete(model).toLong()
    }
```

Android Room