Android : tester l'application

Kotlin

* Logcat
* Présentation

Logcat est un système de journalisation qui fournit un environnement pour la collecte et l’affichage de la sortie de débogage des applications. Les logs incluent des informations sur le déroulement de l’exécution de l’émulateur, c'est-à-dire les erreurs et les messages écrits par l’utilisateur dans l’application via la classe Log. Il est possible de filtrer facilement ces informations grâce à Logcat.

* Vue d'ensemble

Logcat est disponible depuis Android Studio en cliquant sur l'onglet Logcat :



Voici un zoom sur les différentes fonctionnalités offertes par Logcat :



1. Permet de vider les logs.

2. Permet de passer d’une erreur à l’autre.

3. Permet de configurer la présentation des logs.

4. Permet d’afficher les logs selon leur niveau.

5. Permet de faire une recherche dans le contenu des logs.

6. Permet de prédéfinir des filtres de recherche.

Il est important d'utiliser cet outil lors de vos développements. Une bonne utilisation des logs permet de se soustraire de l'utilisation du débogueur.

* Tests unitaires avec JUnit
* Présentation

Les tests unitaires permettent de s'assurer du bon fonctionnement d'une unité de programme. Il est indispensable de mettre en place des tests unitaires tout au long du développement d'une application. JUnit est le célèbre framework permettant de réaliser des tests unitaires.

Par défaut, dans un projet Android, la librairie JUnit est présente dans le fichier de dépendances build.gradle :

dependencies {   
   
//...   
   
    testImplementation 'junit:junit:4.12'   
   
//...   
   
}

Un projet Android a par défaut trois dossiers dans le dossier src :

* Le dossier main contient les sources de l'application.
* Le dossier Test contient les tests unitaires pouvant être exécutés en local.
* Le dossier AndroidTest contient les tests unitaires devant être exécutés sur un appareil mobile ou un émulateur.
* Procédure

Pour créer une classe de test JUnit 4 de base, il est nécessaire de créer une classe Kotlin contenant une ou plusieurs fonctions de test. Une fonction de test commence par l'annotation @Test et contient les instructions permettant de vérifier une fonctionnalité unique.

* Démonstration

Cette démonstration présente une application contenant une base de données et une DAO créées par l'ORM Room (cf. chapitre Android : quelques librairies incontournables - Room). La base de données contient une table personnes.

Personne.kt

package fr.acos.junitwithkotlin.entities   
   
import android.arch.persistence.room.ColumnInfo   
import android.arch.persistence.room.Entity   
import android.arch.persistence.room.PrimaryKey   
   
@Entity(tableName = "personnes")   
data class Personne(   
        @ColumnInfo(name="id") @PrimaryKey(autoGenerate = true) var   
id: Long = 0,   
        @ColumnInfo(name="nom") var nom: String,   
        @ColumnInfo(name="prenom") var prenom: String   
)

Classe représentant la table personnes. Elle est utilisée par l'ORM Room pour construire la table.

PersonneDao.kt

package fr.acos.junitwithkotlin.dao   
   
import android.arch.persistence.room.\*   
import fr.acos.junitwithkotlin.entities.Personne   
   
@Dao   
interface PersonneDao {   
   
    @Query("SELECT \* FROM personnes WHERE nom = :nom")   
    fun get(nom: String): Personne   
   
    @Insert   
    fun insertAll(vararg listCategories: Personne)   
}

Classe permettant de manipuler la table personnes. Les fonctions sont définies par Room.

MaBaseDeDonnees.kt

package fr.acos.junitwithkotlin.dao   
   
import android.arch.persistence.room.Database   
import android.arch.persistence.room.RoomDatabase   
import fr.acos.junitwithkotlin.entities.Personne   
   
   
@Database(entities = [(Personne::class)], version = 1)   
abstract class MaBaseDeDonnees : RoomDatabase()   
{   
    abstract fun personneDao(): PersonneDao   
}

Cette classe représente la base de données et permet de récupérer des objets de type PersonneDao.

PersonneDaoTest.kt

package fr.acos.junitwithkotlin.dao   
   
import android.arch.persistence.room.Room   
import android.support.test.InstrumentationRegistry   
import android.support.test.runner.AndroidJUnit4   
import org.junit.After   
import org.junit.Before   
import org.junit.Test   
import fr.acos.junitwithkotlin.entities.Personne   
import org.junit.Assert.assertEquals   
import org.junit.runner.RunWith   
   
/\*\*   
 \* Classe permettant de tester la classe PersonneDao.   
 \* L'annotation @RunWith(AndroidJUnit4::class) indique que les   
 \* tests doivent être effetués par JUnit.   
 \*/   
@RunWith(AndroidJUnit4::class)   
class PersonneDaoTest   
{   
   
    private var personneDao: PersonneDao? = null   
    private var bdd: MaBaseDeDonnees? = null   
   
    /\*\*   
     \* Fonction exécutée avant tout les tests.   
     \*/   
    @Before   
    fun createDb()   
    {   
        val context = InstrumentationRegistry.getTargetContext()   
        bdd = Room.inMemoryDatabaseBuilder(context,   
MaBaseDeDonnees::class.java!!).build()   
        personneDao = bdd!!.personneDao()   
    }   
   
    /\*\*   
     \* Fonction exécutée après tous les tests   
     \*/   
    @After   
    fun closeDb() {   
        bdd!!.close()   
    }   
   
    /\*\*   
     \* Fonction de test.   
     \*/   
    @Test   
    fun insertAndGetByNameTest()   
    {   
        val user = Personne(0,"Cosson","Barbara")   
        personneDao!!.insertAll(user)   
        val byName = personneDao!!.get("Cosson")   
        assertEquals(user.nom, byName.nom)   
        assertEquals(user.prenom, byName.prenom)   
    }   
}

Cette classe contient les tests unitaires JUnit permettant de tester la classe PersonneDAO. Il est recommandé de tester les bases de données sur un périphérique Android ou sur un émulateur, c'est la raison pour laquelle cette classe de test se trouve dans le dossier AndroidTest.

ExampleInstrumentedTest.kt

package fr.acos.junitwithkotlin   
   
import android.support.test.InstrumentationRegistry   
import android.support.test.runner.AndroidJUnit4   
   
import org.junit.Test   
import org.junit.runner.RunWith   
   
import org.junit.Assert.\*   
   
/\*\*   
 \* Instrumented test, which will execute on an Android device.   
 \*   
 \* See [testing documentation](http://d.android.com/tools/testing).   
 \*/   
@RunWith(AndroidJUnit4::class)   
class ExampleInstrumentedTest {   
    @Test   
    fun useAppContext() {   
        // Context of the app under test.   
        val appContext = InstrumentationRegistry.getTargetContext()   
        assertEquals("fr.acos.junitwithkotlin", appContext.packageName)   
    }   
}

Cette classe est un exemple de test unitaire présent par défaut sur les projets Android.

SuiteDeTests.kt

package fr.acos.junitwithkotlin.suite   
   
import fr.acos.junitwithkotlin.ExampleInstrumentedTest   
import fr.acos.junitwithkotlin.dao.PersonneDaoTest   
import org.junit.runners.Suite   
import org.junit.runner.RunWith   
   
/\*\*   
 \* Classe représentant une suite de tests.   
 \*   
 \* La suite de tests contient les tests des classes de test   
 \* PersonneDaoTest et ExampleInstrumentedTest   
 \*/   
@RunWith(Suite::class)   
@Suite.SuiteClasses(PersonneDaoTest::class,   
ExampleInstrumentedTest::class)   
class SuiteDeTests

Une suite de tests permet de grouper une multitude de tests unitaires dans le but de les exécuter en même temps.

* Test de l'interface avec Espresso
* Présentation

Espresso est une API permettant de tester l’interface d’une application. Il est possible de simuler des actions utilisateur sur l’interface d’une application et ensuite de tester si le résultat d’une action est bien celui attendu, en vérifiant l’état et la valeur d’affichage des différents éléments de l’interface graphique.

Espresso est l’API de base pour tester les interfaces utilisateur ; les dépendances vers l’API Espresso sont déjà présentes dans Gradle au moment de la création de nouveaux projets sous Android Studio. Cette API est appréciée pour sa fiabilité et sa facilité d’utilisation.

Les objectifs de ce chapitre sont les suivants :

* savoir mettre en place Espresso,
* découvrir le fonctionnement de ce type de test.
* Procédure de mise en place

Pour mettre en place Espresso dans un projet, il est nécessaire de suivre les étapes suivantes :

* définir les dépendances nécessaires,
* définir les tests.
* Dépendances

Voici les dépendances nécessaires pour l'utilisation d'Espresso :

dependencies    
   
{   
   
    ...   
   
    androidTestImplementation   
'com.android.support.test.espresso:espresso-core:3.0.2'   
   
    androidTestImplementation 'com.android.support.test:   
runner:1.0.2'   
   
    androidTestImplementation   
'com.android.support.test:rules:1.0.2'   
   
    ...   
   
}

* Définir un test
* Définir une classe de test

Pour définir un test sur une activité avec Espresso, il est tout d’abord nécessaire de créer une classe de test dans l’arborescence du dossier AndroidTest :

MainActivityTest.kt

@LargeTest   
   
class MainActivityTest   
   
{   
   
   //...   
   
}

* Définir l'activité à tester dans la classe de test

Il est nécessaire de définir une JUnit Rule dans la classe de test pour définir l'activité à tester.

MainActivityTest.kt

@LargeTest   
   
class MainActivityTest   
   
{   
   
   @Rule @JvmField val activity = ActivityTestRule<MainActivity>(MainActivity::class.java)   
   
}

* Définir un cas de test

Un cas de test est un scénario décrivant l'action d'un utilisateur avec l'application. Une classe de test doit contenir plusieurs cas de test afin de couvrir le maximum de scénarios possibles.

MainActivityTest.kt

class MainActivityTest   
   
{   
   
    @Rule @JvmField val activity =   
ActivityTestRule<MainActivity>(MainActivity::class.java)   
   
   
   
    @Test   
   
    fun testRandom()   
   
    {   
   
        //Clic sur l'élément dont l'id est égal à 'R.id.btn\_bonjour'   
   
        onView(withId(R.id.btn\_bonjour)).perform(click())   
   
        //Regarde si l'élément dont l'id est égal à 'R.id.tv\_affichage' est   
        //égal à 'Bonjour'   
   
        onView(withId(R.id.tv\_affichage)).check(matches(withText("Bonjour")))   
   
    }

* Démonstration

Cette démonstration présente comment tester une interface graphique simple.

Activity\_main.xml

<?xml version="1.0" encoding="utf-8"?>   
<android.support.constraint.ConstraintLayout xmlns:android=  
"http://schemas.android.com/apk/res/android"   
    xmlns:app="http://schemas.android.com/apk/res-auto"   
    xmlns:tools="http://schemas.android.com/tools"   
    android:layout\_width="match\_parent"   
    android:layout\_height="match\_parent"   
    tools:context="fr.acos.espressowithkotlin.MainActivity">   
   
    <TextView   
        android:id="@+id/tv\_affichage"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Hello World!"   
        app:layout\_constraintBottom\_toBottomOf="parent"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toTopOf="parent" />   
   
    <Button   
        android:id="@+id/btn\_bonjour"   
        android:layout\_width="wrap\_content"   
        android:layout\_height="wrap\_content"   
        android:text="Bonjour"   
        app:layout\_constraintLeft\_toLeftOf="parent"   
        app:layout\_constraintRight\_toRightOf="parent"   
        app:layout\_constraintTop\_toBottomOf="@id/tv\_affichage"   
        android:onClick="onClickButton"   
        />   
   
</android.support.constraint.ConstraintLayout>

Dans cet exemple, l'IHM contient :

* Une TextView qui affiche "Bonjour" lorsqu'il y a un clic sur le bouton "Bonjour".
* Un bouton "Bonjour" qui permet d'afficher "Bonjour" dans la TextView.

MainActivity.kt

package fr.acos.espressowithkotlin   
   
import android.support.v7.app.AppCompatActivity   
import android.os.Bundle   
import android.view.View   
import kotlinx.android.synthetic.main.activity\_main.\*   
   
class MainActivity : AppCompatActivity() {   
   
    override fun onCreate(savedInstanceState: Bundle?)   
    {   
        super.onCreate(savedInstanceState)   
        setContentView(R.layout.activity\_main)   
    }   
   
    /\*\*   
     \* Fonction qui affiche "Bonjour" lorsqu'il y a un clic sur le bouton   
     \*/   
    fun onClickButton(view:View)   
    {   
        tv\_affichage.text = "Bonjour"   
    }   
}

Dans cet exemple, le contrôleur contient : une fonction onClickButton() qui exécute une instruction permettant de changer le contenu de la TextView tv\_affichage.

La classe MainActivityTest doit être créée dans le dossier AndroidTest.

MainActivityTest.kt

package fr.acos.espressowithkotlin   
   
import android.support.test.espresso.Espresso.onView   
import android.support.test.espresso.action.ViewActions.click   
import android.support.test.espresso.assertion.ViewAssertions.matches   
import android.support.test.espresso.matcher.ViewMatchers.withId   
import android.support.test.espresso.matcher.ViewMatchers.withText   
import android.support.test.filters.LargeTest   
import android.support.test.rule.ActivityTestRule   
import android.support.test.runner.AndroidJUnit4   
import org.junit.Rule   
import org.junit.Test   
import org.junit.runner.RunWith   
   
   
/\*\*   
 \* Classe de test   
 \*/   
class MainActivityTest   
{   
    /\*\*   
     \* Permet de définir l'activité à tester   
     \*/   
    @Rule @JvmField val activity =   
ActivityTestRule<MainActivity>(MainActivity::class.java)   
   
    /\*\*   
     \* Fonction qui définit un test.   
     \*/   
    @Test   
    fun testAction()   
    {   
        //Clic sur le bouton "Bonjour"   
        onView(withId(R.id.btn\_bonjour)).perform(click())   
        //Regarde si l'affichage est bien égal à "Bonjour"   
        onView(withId(R.id.tv\_affichage)).check(matches(withText("Bonjour")))   
    }   
}

Cette classe de test permet de vérifier que la TextView affiche bien "Bonjour" lorsqu'il y a un clic sur le bouton.