République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Benyoucef BENKHEDDA -Alger1-

Faculté des Sciences

Département Informatique

Rapport

Projet Développement Mobile Avancés

Construction d'une application question choix multiple 'QCM' appelé 'QuizApp'

Réalisé par

AOUNE Ramzi

GOUIZI Mounia

GHEZAL Ali Mehdi

SALMANE Imad Eddine

2023/2024

I. Introduction :

Le développement d'applications mobiles a pris une place prépondérante dans le paysage technologique contemporain, et parmi ces créations, l'application Quiz App se distingue par son ingéniosité et son interface utilisateur captivante. Conçue pour offrir une expérience de quiz interactive et enrichissante, cette application a été élaborée avec soin, tant du point de vue du code que de l'expérience utilisateur.

Comme si on a dans chaque application sa structure de document et du programme constitué de deux parties, la première représente la structure de document XML qui définit l'interface de l'application avec ses divers codes, après on a la deuxième partie qui représente notre structure de programme nécessaire en JAVA dont notre aventure d'invention du notre projet Quiz App commence.

L'aventure commence par une exploration des différentes facettes de l'application, avec une attention particulière portée aux fichiers principaux qui composent le cœur de Quiz App. Ces fichiers, regroupés sous les titres "**main**", "**androidTest**", et "**test**"

Le chapitre principal, "**main**", expose les composants clés de l'application, de GameWon à HomeScreen, chacun jouant un rôle essentiel dans la fluidité et la convivialité de l'expérience utilisateur. Les sous-titres de cette section détaillent les codes cruciaux, dévoilant ainsi les coulisses de la magie opérée par l'application.

Le volet "**androidTest**" plonge dans le domaine de la vérification, s'assurant que le contexte de l'application est correct. Un seul fichier, ExampleInstrumentedTest.java, se distingue ici, mais son rôle est crucial pour garantir la fiabilité de l'application dans divers scénarios.

Enfin, la section "**test**" s'attarde sur les fondamentaux avec ExampleUnitTest.java. Cet aspect du développement met à l'épreuve les opérations arithmétiques de base, assurant ainsi la robustesse des fonctionnalités clés.

Cette exploration approfondie sera guidée par des captures de codes essentiels, chacune suivie d'une description éclairante. La conclusion, quant à elle, réunira les fils conducteurs de cette aventure dans une synthèse révélatrice de l'excellence de la Quiz App. Bienvenue dans un voyage au cœur du code et de la conception Android.

II. Structure du document « XML » :

Le document XML représente un ensemble des codes contient des balises à utiliser pour qu'on a construit notre interface visuelle de l'application Quiz App avant de mettre en œuvre le fonctionnement de l'application via les programmes java. Dans ce document, on distingue 8 sous document XML ce sont : **'AndroidManifest', 'assetWizardSettings', 'compiler', 'deploymentTargetDropDown', 'gradle', 'migration', 'misc',** et **'workspace'**

II.1. *AndroidManifest :*

```
5       <application
6           android:allowBackup="true"
7           android:dataExtractionRules="@xml/data_extraction_rules"
8           android:fullBackupContent="@xml/backup_rules"
9           android:icon="@mipmap/ic_launcher"
10          android:label="@string/app_name"
11          android:roundIcon="@mipmap/ic_launcher_round"
12          android:supportsRtl="true"
13          android:theme="@style/Theme.QuizGame"
14          tools:targetApi="31">
```

Cette capture contient les paramètres de base liés à l'application, tels que la configuration de l'icône, l'étiquette, l'icône ronde et le thème.

```
16          <activity
17              android:name=".SplashScreenActivity"
18              android:exported="true">
19              <intent-filter>
20                  <action android:name="android.intent.action.MAIN" />
21
22                  <category android:name="android.intent.category.LAUNCHER" />
23              </intent-filter>
24          </activity>
```

Cette capture concerne la configuration de l'activité liée à l'écran de démarrage. L'activité principale **'SplashScreenActivity'** est définie comme une activité exportée et déclarée comme l'activité de lancement principal avec une intention de lancement.

II.2. *assetWizardSettings :*

```
<entry key="actionbar">
  <value>
    <PersistentState>
      <option name="children">
        <map>
          <entry key="clipArt">
            <value>
              <PersistentState>
                <option name="values">
                  <map>
                    <entry key="color" value="000000" />
                    <entry key="imagePath" value="C:\Users\gazty\AppData\Local\Temp\ic_android_black_24dp.xml
                  </map>
                </option>
              </PersistentState>
            </value>
          </entry>
          <entry key="text">
            <value>
              <PersistentState>
                <option name="values">
                  <map>
                    <entry key="color" value="000000" />
                  </map>
```

Ce code XML représente la configuration des paramètres du wizard pour la gestion des images. La capture 1 se concentre sur les paramètres liés à **'imageWizard**', **'actionbar**', et '**clipArt**', avec des valeurs spécifiques pour la couleur et le chemin de l'image.

```
<entry key="text">
  <value>
    <PersistentState>
      <option name="values">
        <map>
          <entry key="color" value="000000" />
        </map>
      </option>
    </PersistentState>
  </value>
</entry>
```

Cette capture 2 concerne les paramètres spécifiques liés à '**text**' sous '**actionbar**', détaillant la couleur associée.

II.3. *compiler :*

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="CompilerConfiguration">
    <bytecodeTargetLevel target="17" />
  </component>
</project>
```

Le code XML ci-dessus représente la configuration du compilateur avec le paramètre '**bytecodeTargetLevel**'. La capture se concentre sur ce paramètre, indiquant un niveau de bytecode cible de 17.

II.4. *deploymentTargetDropDown :*

```
<targetSelectedWithDropDown>
  <Target>
    <type value="QUICK_BOOT_TARGET" />
    <deviceKey>
      <Key>
        <type value="VIRTUAL_DEVICE_PATH" />
        <value value="C:\Users\gazty\.android\avd\Pixel_3a_API_34_extension_level_7_x86_64.avd" />
      </Key>
    </deviceKey>
  </Target>
</targetSelectedWithDropDown>
```

Le code XML ci-dessus représente la configuration des cibles de déploiement avec le sous-titre '**deploymentTargetDropDown**'. La capture se concentre sur les paramètres liés à '**deploymentTargetDropDown**', décrivant la sélection d'une cible avec les détails du périphérique virtuel.

II.5. *gradle :*

```
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="GradleMigrationSettings" migrationVersion="1" />
  <component name="GradleSettings">
    <option name="linkedExternalProjectsSettings">
      <GradleProjectSettings>
        <option name="externalProjectPath" value="$PROJECT_DIR$" />
        <option name="gradleJvm" value="#GRADLE_LOCAL_JAVA_HOME" />
        <option name="modules">
          <set>
            <option value="$PROJECT_DIR$" />
            <option value="$PROJECT_DIR$/app" />
          </set>
        </option>
        <option name="resolveExternalAnnotations" value="false" />
      </GradleProjectSettings>
    </option>
  </component>
</project>
```

Le code XML ci-dessus représente la configuration Gradle avec le sous-titre 'gradle.xml'. La capture se concentre sur les paramètres liés à '**GradleSettings**' et inclut des informations telles que le chemin du projet externe, la version de JVM Gradle, et les modules associés.

II.6. *migration :*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="ProjectMigrations">
    <option name="MigrateToGradleLocalJavaHome">
      <set>
        <option value="$PROJECT_DIR$" />
      </set>
    </option>
  </component>
</project>
```

Cette capture englobe l'en-tête du fichier XML et se concentre sur les paramètres globaux liés à '**ProjectMigrations**', notamment la migration vers le répertoire local de Java pour Gradle, avec le chemin du projet externe.

II.7. *misc :*

```xml
<project version="4">
  <component name="ExternalStorageConfigurationManager" enabled="true" />
  <component name="ProjectRootManager" version="2" languageLevel="JDK_17" default="true" project-jdk-name="jbr-17" project-jdk-type="JavaSDK">
    <output url="file://$PROJECT_DIR$/build/classes" />
  </component>
  <component name="ProjectType">
    <option name="id" value="Android" />
  </component>
</project>
```

Cette capture englobe l'en-tête du fichier XML et se concentre sur les paramètres globaux liés à '**ExternalStorageConfigurationManager**', '**ProjectRootManager**' et '**ProjectType**' avec la spécification de l'ID du type de projet (Android). (CAP2)

II.8. *workspace :*

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project version="4">
  <component name="AndroidLayouts">
    <shared>
      <config />
    </shared>
    <layouts>
      <layout url="file://$PROJECT_DIR$/app/src/main/res/layout-large/activity_game_main.xml">
        <config>
          <theme>@style/Theme.QuizGame</theme>
        </config>
      </layout>
      <layout url="file://$PROJECT_DIR$/app/src/main/res/layout-large/activity_home_screen.xml">
        <config>
          <theme>@style/Theme.QuizGame</theme>
        </config>
      </layout>
      <layout url="file://$PROJECT_DIR$/app/src/main/res/layout-large/activity_time__up.xml">
        <config>
          <theme>@style/Theme.QuizGame</theme>
        </config>
      </layout>
```

Le code XML ci-dessus représente la configuration du projet avec le sous-titre '**workspace.xml'**. La capture se concentre sur le composant '**AndroidLayouts**' avec ses paramètres spécifiques, notamment la spécification du thème pour différents layouts du projet, tels que 'activity_game_main.xml', 'activity_home_screen.xml', etc. Ces paramètres contribuent à la configuration visuelle de l'interface utilisateur pour différentes tailles d'écrans et orientations.

III. Structure du programme « Java »:

Le programme est structuré en plusieurs classes pour promouvoir la modularité et la lisibilité du code, on distingue 3 grandes dossiers java qui contient plusieurs classes à implémenter, ce sont : **‘Main’**, **‘AndroidTest’** et **‘Test’**

1. Main :

C’est le dossier principal qui a plusieurs classes java à implémenter, dont on a besoin de 8 classes, chaque classe à son propre code destiné à plusieurs rôles, ces 8 classes sont : **‘GameWon’, ‘HomeSection’, ‘MainGameActivity’, ‘PlayAgain’, ‘SplashScreenActivity’, ‘Time_Up’, ‘TriviaQuestion’,** et **‘TriviaQuizHelper’**

1.1. *GameWon:*

```
18      public void PlayAgain(View view) {
19          Intent intent = new Intent(GameWon.this, MainGameActivity.class);
20          startActivity(intent);
21          finish();
22      }
23  }
```

Ce code extrait le nouveau intent qui a déclenché l'activité **GameWon** et l'affiche par le déclarement de la fonction **‘startActivity’** correspondant. Il illustre le transfert de données entre les activités et la mise à jour de l'interface utilisateur en conséquence.

```
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.game_won);
```

Cette capture dévoile le gestionnaire de sauvegarde par la méthode **‘super.onCreate**’, il lance une nouvelle instance de **setContentView**, offrant ainsi à l'utilisateur de visualiser le résultat **game_won**.

```
8   public class GameWon extends Activity {
9
10      @Override
11      protected void onCreate(Bundle savedInstanceState) {
12          super.onCreate(savedInstanceState);
13          setContentView(R.layout.game_won);
14      }
15
```

Cette portion de code gère la création de l’instance **‘OnCreate**’

1.2. *HomeSection:*

```
25            //PlayGame button - it will take you to the MainGameActivity
26            playGame.setOnClickListener(new View.OnClickListener() {
27                @Override
28                public void onClick(View view) {
29                    Intent intent = new Intent(HomeScreen.this,MainGameActivity.class);
30                    startActivity(intent);
31                    finish();
32                }
33            });
```

Ce code gère le bouton "**Start**" dans l'écran d'accueil **(HomeScreen).** Lorsque le bouton est pressé grâce à la fonction void '**OnClick'**, il crée un intent pour lancer l'activité **MainGameActivity**, marquant ainsi le début du jeu. La méthode **finish()** garantit que l'écran d'accueil est fermé après le passage à l'activité suivante.

```
35            //Quit button - This will quit the game
36            quit.setOnClickListener(new View.OnClickListener() {
37                @Override
38                public void onClick(View view) {
39                    finish();
40                }
41            });
42        }
```

Cette partie du code gère le bouton "**Quit**" dans l'écran d'accueil. En appuyant sur ce bouton, l'application se termine. Il s'agit d'une fonctionnalité essentielle pour permettre aux utilisateurs de quitter l'application facilement.

```
13  public class HomeScreen extends AppCompatActivity {
14      FButton playGame,quit;
15      TextView tQ;
16
17      @Override
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_home_screen);
21
22          //the below method will initialize views
23          initViews();
24
25          //PlayGame button - it will take you to the MainGameActivity
26          playGame.setOnClickListener(new View.OnClickListener() {
27              @Override
28              public void onClick(View view) {
29                  Intent intent = new Intent(HomeScreen.this,MainGameActivity.class);
30                  startActivity(intent);
31                  finish();
32              }
33          });
```

Cette capture révèle le gestionnaire d'événements pour le bouton "**Start**". Lorsqu'il est pressé, il crée un intent pour ouvrir l'activité '**MainGameActivity'**, où le menu sera affiché.

1.3. *MainGameActivity:*

```
129     public void updateQueAndOptions() {
130
131         //This method will setText for que and options
132         questionText.setText(currentQuestion.getQuestion());
133         buttonA.setText(currentQuestion.getOptA());
134         buttonB.setText(currentQuestion.getOptB());
135         buttonC.setText(currentQuestion.getOptC());
136         buttonD.setText(currentQuestion.getOptD());
137
138         timeValue = 20;
139
140         //Now since the user has ans correct just reset timer back for another que- by cancel and start
141         countDownTimer.cancel();
142         countDownTimer.start();
143
144         //set the value of coin text
145         coinText.setText(String.valueOf(coinValue));
146         //Now since user has ans correct increment the coinvalue
147         coinValue++;
148
149     }
```

La méthode **updateQueAndOptions()** est essentielle dans **MainGameActivity**. Elle gère l'affichage de la question et les options pendant le jeu. Cette fonction est appelée chaque fois qu'une nouvelle question doit être présentée aux utilisateurs.

```
312         buttonNext.setOnClickListener(new View.OnClickListener() {
313             @Override
314             public void onClick(View view) {
315                 //This will dismiss the dialog
316                 dialogCorrect.dismiss();
317                 //it will increment the question number
318                 qid++;
319                 //get the que and 4 option and store in the currentQuestion
320                 currentQuestion = list.get(qid);
321
322                 //Now this method will set the new que and 4 options
323                 updateQueAndOptions();
324                 //reset the color of buttons back to white
325                 resetColor();
326                 //Enable button - remember we had disable them when user ans was correct in there particular button methods
327                 enableButton();
328             }
329         });
```

La fonction **setOnClickListener()** est responsable de la mise à jour des questions et des options. Elle prend un paramètre de questions, puis effectue les opérations nécessaires pour mettre à jour les couleurs et l'activation des boutons dans le jeu.

```
235     public void gameLostPlayAgain() {
236
237         Intent intent = new Intent(this, PlayAgain.class);
238         int coin = coinValue - 1;
239         String s = currentQuestion.getAnswer();
240
241         String score = String.valueOf(coin);
242         intent.putExtra("yourScore", score);
243         intent.putExtra("CorrectAns", s);
244
245         startActivity(intent);
246         finish();
247     }
```

La méthode **gameLostPlayAgain()** est appelée lorsque le jeu se termine. Elle contient la logique nécessaire pour afficher les résultats du joueur, tel que la réponse correcte après que l'utilisateur joueur a choisi la réponse érroné.

1.4. *PlayAgain:*

```
21      protected void onCreate(Bundle savedInstanceState) {
22          super.onCreate(savedInstanceState);
23          setContentView(R.layout.play_again);
24          Intent intent = getIntent();
25          if (intent != null) {
26              String receivedString = intent.getStringExtra("yourScore");
27              String correctAns = intent.getStringExtra("CorrectAns");
28
29              if (receivedString != null) {
30                  // Find the TextView in your layout
31                  TextView textViewContent = findViewById(R.id.textViewScore);
32
33                  // Set the received string to the TextView
34                  textViewContent.setText(receivedString);
35              }
```

La méthode **onCreate()** de **PlayAgain.java** est chargée de configurer l'interface utilisateur lorsque l'activité est créée. Elle gère également l'affichage du score du joueur et des réponses correctes.

```
55          playAgain.setOnClickListener(new View.OnClickListener() {
56              @Override
57              public void onClick(View view) {
58                  Intent intent = new Intent(PlayAgain.this, MainGameActivity.class);
59                  startActivity(intent);
60                  finish();
61              }
62          });
63          exit.setOnClickListener(new View.OnClickListener() {
64              @Override
65              public void onClick(View view) {
66                  Intent intent = new Intent(PlayAgain.this, HomeScreen.class);
67                  startActivity(intent);
68                  finish();
69              }
70          });
```

L'OnClickListener de **playAgainButton** est configuré pour redémarrer le jeu en créant une nouvelle instance de **MainGameActivity** lorsqu'il est cliqué.

```
86        public void onBackPressed() {
87            finish();
88        }
```

La méthode **onBackPressed()** est surchargée pour gérer le comportement lorsque l'utilisateur appuie sur le bouton de retour. Dans ce cas, l'activité actuelle est simplement terminée.

1.5. *SplashScreenActivity:*

```
19      protected void onCreate(Bundle savedInstanceState) {
20          super.onCreate(savedInstanceState);
21          setContentView(R.layout.activity_splash_screen);
22          logoImageView = findViewById(R.id.logo);
23          ObjectAnimator scaleX = ObjectAnimator.ofFloat(logoImageView, "scaleX", 1.0f, 5.0f);
24          ObjectAnimator scaleY = ObjectAnimator.ofFloat(logoImageView, "scaleY", 1.0f, 5.0f);
25          scaleX.setInterpolator(new AccelerateDecelerateInterpolator());
26          scaleY.setInterpolator(new AccelerateDecelerateInterpolator());
27
28          scaleX.setDuration(SPLASH_DURATION);
29          scaleY.setDuration(SPLASH_DURATION);
30
31          scaleX.start();
32          scaleY.start();
```

La méthode **onCreate()** de **SplashScreenActivity** configure l'interface utilisateur de l'écran de démarrage et utilise des animations pour le logo. Elle dirige ensuite l'utilisateur vers l'écran d'accueil après la durée spécifiée.

```
15  public class SplashScreenActivity extends AppCompatActivity {
16      private static final int SPLASH_DURATION = 2000;
17      private ImageView logoImageView;
18      @Override
```

Cette capture de code présente les déclarations de variables et constantes importantes utilisées dans **SplashScreenActivity.java.**

```
41          new Handler().postDelayed(new Runnable() {
42              @Override
43              public void run() {
44                  startActivity(new Intent(SplashScreenActivity.this, HomeScreen.class));
45                  finish();
46              }
47          }, SPLASH_DURATION + 500);
48      }
49  }
```

La méthode **onBackPressed()** est surchargée pour gérer le comportement lorsque l'utilisateur appuie sur le bouton de retour pendant l'écran de démarrage. Dans ce cas, l'activité actuelle est simplement terminée

1.6. *Time_Up :*

```
18      protected void onCreate(Bundle savedInstanceState) {
19          super.onCreate(savedInstanceState);
20          setContentView(R.layout.activity_time__up);
21          //Initialize
22          playAgainButton = (FButton)findViewById(R.id.playAgainButton);
23          timeUpText = (TextView)findViewById(R.id.timeUpText);
```

La méthode **onCreate()** de **Time_Up** initialise les éléments de l'interface utilisateur, configure un gestionnaire d'événements pour le bouton "**Play Again**" et applique une police personnalisée aux textes.

```
26          playAgainButton.setOnClickListener(new View.OnClickListener() {
27              @Override
28              public void onClick(View view) {
29                  Intent intent = new Intent(Time_Up.this,MainGameActivity.class);
30                  startActivity(intent);
31                  finish();
32
33
34              }
```

La méthode **onBackPressed()** de **Time_Up** est surchargée pour terminer l'activité si l'utilisateur appuie sur le bouton de retour pendant l'écran de fin de temps.

```
52      public void onBackPressed() {
53          super.onBackPressed();
54          finish();
55      }
56  }
```

Cette capture de code montre la configuration du bouton "**Play Again**" avec un gestionnaire d'événements et l'application d'une police personnalisée aux éléments de l'interface utilisateur. Elle illustre également la transition vers **MainGameActivity** lors du clic sur le bouton.

1.7. *TriviaQuestion :*

```
14      public TriviaQuestion(String q, String oa, String ob, String oc, String od, String ans) {
15
16          question = q;
17          opta = oa;
18          optb = ob;
19          optc = oc;
20          optd = od;
21          answer = ans;
22      }
```

Le constructeur de la classe **TriviaQuestion** prend en paramètre les éléments d'une question (texte de la question, options A, B, C, D, et la réponse) et initialise les propriétés correspondantes.

```
34      public String getQuestion() {
35          return question;
36      }
```

La méthode **getQuestion()** de **TriviaQuestion** renvoie le texte de la question.

```
82      public void setAnswer(String ans) {
83          answer = ans;
84      }
```

La méthode **setAnswer(String ans)** de **TriviaQuestion** permet de définir la réponse à la question.

1.8. *TriviaQuizHelper :*

```
39      private static final String CREATE_TABLE = "CREATE TABLE " + TABLE_NAME + " ( " + UID + " INTEGER PRIMARY KEY AUTOINCREMENT , " + QUE
```

Cette constante **CREATE_TABLE** représente la requête SQL pour la création de la table dans la base de données. Elle définit la structure des colonnes, y compris l'ID, la question, les options A, B, C, D et la réponse.

```
109     private void addAllQuestions(ArrayList<TriviaQuestion> allQuestions) {
110         SQLiteDatabase db = this.getWritableDatabase();
111         db.beginTransaction();
112         try {
113             ContentValues values = new ContentValues();
114             for (TriviaQuestion question : allQuestions) {
115                 values.put(QUESTION, question.getQuestion());
116                 values.put(OPTA, question.getOptA());
117                 values.put(OPTB, question.getOptB());
118                 values.put(OPTC, question.getOptC());
119                 values.put(OPTD, question.getOptD());
120                 values.put(ANSWER, question.getAnswer());
121                 db.insert(TABLE_NAME, null, values);
122             }
123             db.setTransactionSuccessful();
124         } finally {
125             db.endTransaction();
126             db.close();
127         }
128     }
```

La méthode **addAllQuestions(ArrayList<TriviaQuestion> allQuestions)** ajoute toutes les questions fournies à la base de données. Elle utilise **ContentValues** pour insérer les données dans la table.

```
131     List<TriviaQuestion> getAllOfTheQuestions() {
132
133         List<TriviaQuestion> questionsList = new ArrayList<>();
134         SQLiteDatabase db = this.getWritableDatabase();
135         db.beginTransaction();
136         String coloumn[] = {UID, QUESTION, OPTA, OPTB, OPTC, OPTD, ANSWER};
137         Cursor cursor = db.query(TABLE_NAME, coloumn, null, null, null, null, null);
138
139
140         while (cursor.moveToNext()) {
141             TriviaQuestion question = new TriviaQuestion();
142             question.setId(cursor.getInt(0));
143             question.setQuestion(cursor.getString(1));
144             question.setOptA(cursor.getString(2));
145             question.setOptB(cursor.getString(3));
146             question.setOptC(cursor.getString(4));
147             question.setOptD(cursor.getString(5));
148             question.setAnswer(cursor.getString(6));
149             questionsList.add(question);
150         }
151
152         db.setTransactionSuccessful();
153         db.endTransaction();
154         cursor.close();
```

La méthode **getAllOfTheQuestions()** de **TriviaQuizHelper** récupère toutes les questions de la base de données et les renvoie sous forme de liste d'objets **TriviaQuestion**. Elle utilise **Cursor** pour parcourir les résultats de la requête **SELECT**.

2. AndroidTest :

C’est le deuxième dossier contient une seule classe à manipuler, nommé la classe : **‘ExampleInstrumentedTest’**

ExampleInstrumentedTest :

```java
package com.example.quiz_game;

import android.content.Context;

import androidx.test.platform.app.InstrumentationRegistry;
import androidx.test.ext.junit.runners.AndroidJUnit4;

import org.junit.Test;
import org.junit.runner.RunWith;

import static org.junit.Assert.*;

/**
 * Instrumented test, which will execute on an Android device.
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
@RunWith(AndroidJUnit4.class)
public class ExampleInstrumentedTest {
    @Test
    public void useAppContext() {
        // Context of the app under test.
        Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
        assertEquals("com.example.quiz_game", appContext.getPackageName());
    }
}
```

Cette classe **ExampleInstrumentedTest** est un test instrumenté qui vérifie si le contexte de l'application a le bon nom de package. Il utilise la classe **InstrumentationRegistry** pour accéder au contexte de l'application en cours de test et compare le nom du package avec **"com.example.quiz_game"**. Ce type de test est crucial pour s'assurer que le contexte de l'application fonctionne correctement dans l'environnement Android réel.

3. Test :

C’est le troisième dossier contient une seule classe à manipuler, nommé la classe : **‘ExampleUnitTest’**

ExampleUnitTest :

```java
package com.example.quiz_game;

import org.junit.Test;

import static org.junit.Assert.*;

/**
 * Example local unit test, which will execute on the development machine (host).
 *
 * @see <a href="http://d.android.com/tools/testing">Testing documentation</a>
 */
public class ExampleUnitTest {
    @Test
    public void addition_isCorrect() {
        assertEquals(4, 2 + 2);
    }
}
```

Cette classe **ExampleUnitTest** est un exemple de test unitaire qui vérifie si l'opération d'addition de deux nombres fonctionne correctement. Dans ce cas, le test s'assure que l'addition de 2 et 2 est égale à 4. Les tests unitaires sont essentiels pour garantir le bon fonctionnement des fonctions de base de l'application et identifier tout comportement indésirable.

IV. Conclusion :

En conclusion, le développement de l'application **Quiz_Game** a été une expérience enrichissante et éducative. À travers ce projet, nous avons exploré divers aspects du développement d'applications Android, en mettant en œuvre des fonctionnalités telles que le jeu de questions-réponses, les écrans de jeu, les écrans de résultats et plus encore.

L'architecture du code a été soigneusement structurée, avec des classes dédiées à des fonctions spécifiques. La classe **MainGameActivity** a servi de pivot principal pour la logique de jeu, tandis que d'autres classes telles que **PlayAgain**, **SplashScreenActivity**, et **Time_Up** ont contribué à des fonctionnalités spécifiques et à l'expérience utilisateur.

La gestion des questions du quiz a été réalisée avec la classe **TriviaQuizHelper**, qui utilise une base de données SQLite pour stocker les questions, leurs options et les bonnes réponses. Cela permet une extensibilité facile pour ajouter davantage de questions au jeu.

Les tests, bien que limités dans cet exemple, sont cruciaux pour garantir la fiabilité et la stabilité de l'application. La classe **ExampleUnitTest** montre comment un simple test unitaire peut être mis en œuvre pour vérifier le bon fonctionnement de certaines fonctionnalités.

En termes d'améliorations futures, l'ajout de fonctionnalités supplémentaires, de niveaux de difficulté ou même d'une interface utilisateur plus élaborée pourrait être envisagé. L'optimisation des performances et la prise en charge de différents appareils Android pourraient également être des points d'amélioration.

En résumé, le projet Quiz_Game illustre le processus de développement d'une application Android interactive, divertissante et éducative. Il a fourni une base solide pour comprendre les concepts clés du développement Android et a ouvert la porte à des améliorations futures passionnantes.