ANDROID GK QUIZ

Mobile Application Development

UNIVERSITY of TECHNOLOGY, MAURITIUS

Name: BEEHARRY Mohammad Taajuddin (2003_19375)

BSC (HONS) SOFTWARE ENGINEERING

Cohort: BSE20AFT

Year 2 Semester 2

Lecturer: Dr. G. Suddul

Table of Contents

List of Figures

GK Quiz Design

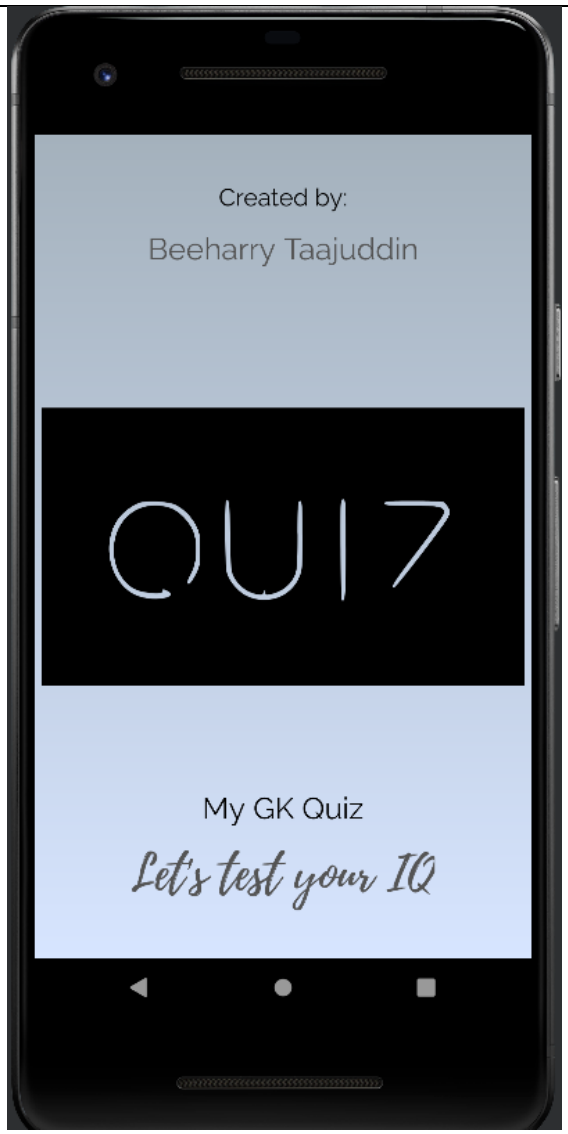


Figure 1 - Splash Screen

Figure 1 depicts the splash screen of GK Quiz for 3 seconds.

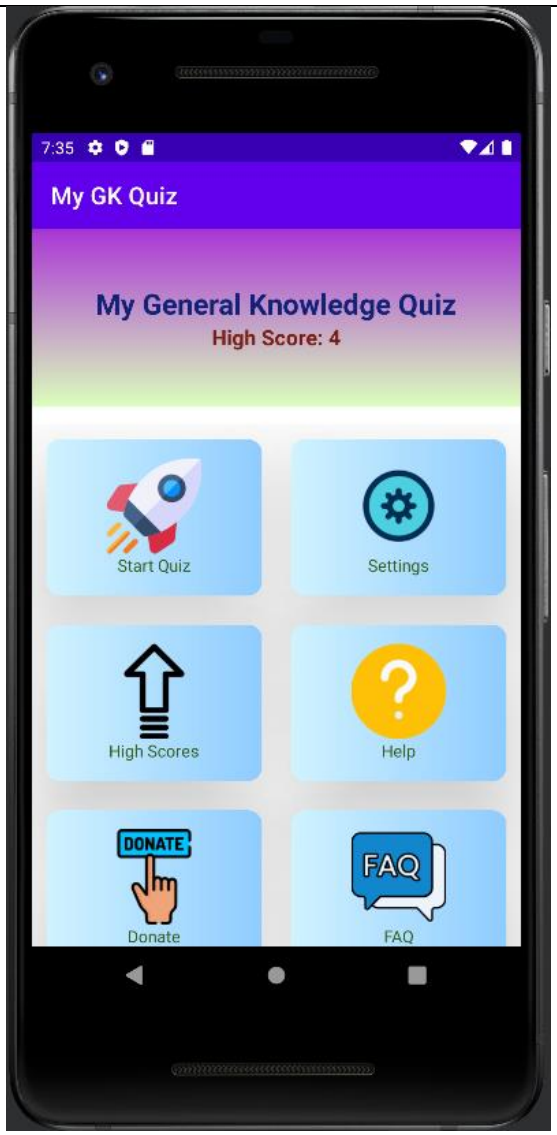


Figure 2 – Dashboard

Figure 2 depicts the main page where it contains the image buttons and form which a user can check his high score.

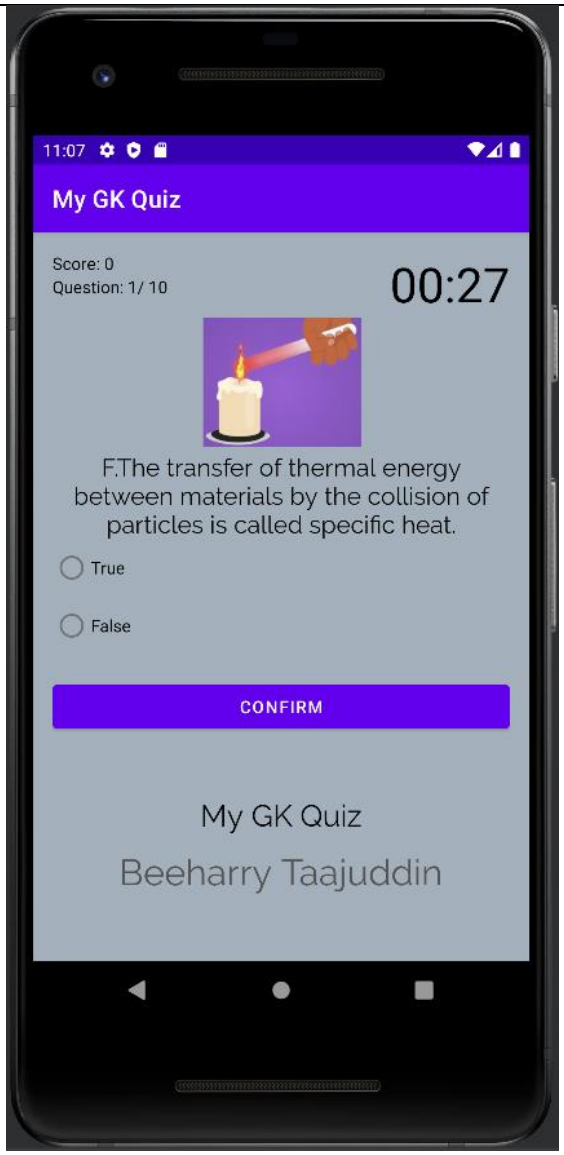


Figure 3 - Quiz Started Ques1

Figure 3 shows that the question one has been displayed. Radio button has been used to represent true and false. Each question will be answered in 30 seconds by the user and also that a countdown has been implemented to prevent the user from cheating. The score and the number of questions are displayed above so that the user aware of how many questions left and what's his score.

Figure 4 - Countdown End

Figure 4 demonstrates that the user has selected an answer but has not pressed the *'confirm'* button. As a result, the countdown will continue until **00:00**. When the timer runs out, the button will change to *'next'* and the correct will be presented in the same place where the question was displayed. Moreover, after the countdown end, it checks to see if the user has provided the correct answer. If yes, the score will increment by 1 as indicated in figure 4 at the top-right corner.

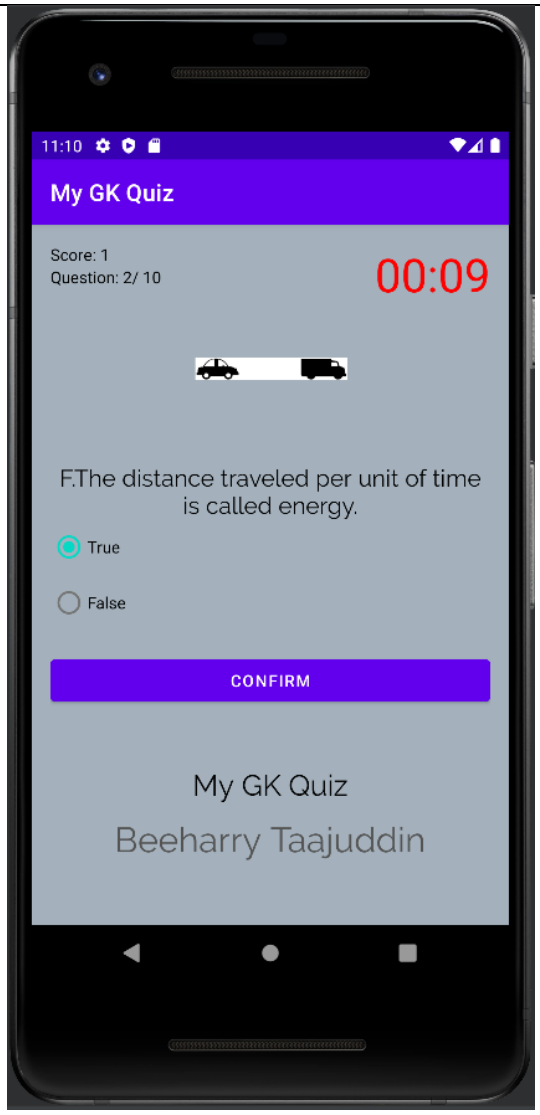


Figure 5 - Countdown below 10s

Figure 5 shows that when the reaches 10 seconds, the text color is changed to red so that it creates an expression for the user to wrap up with the question.

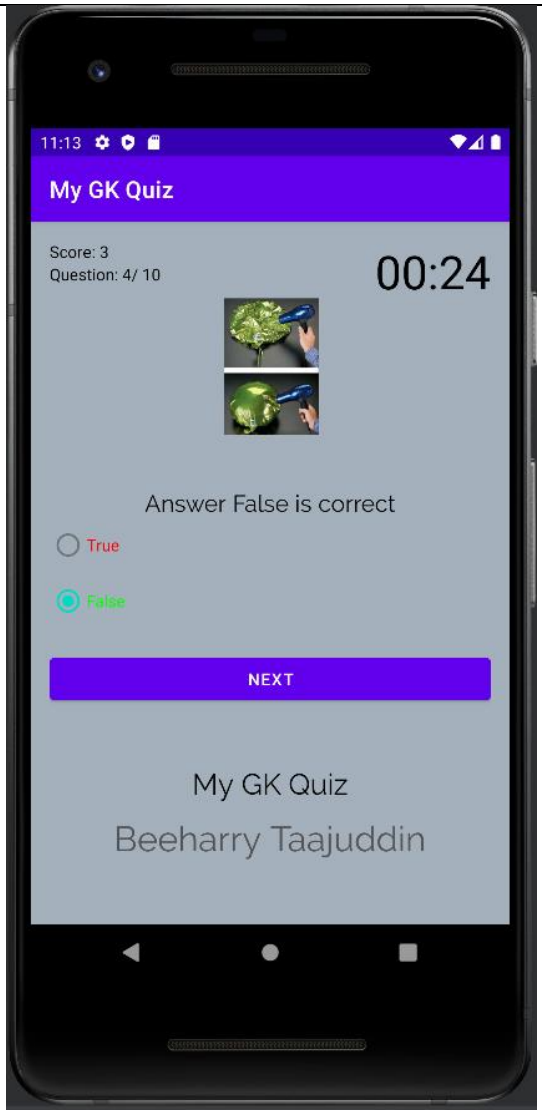


Figure 6 – Countdown freeze when confirming answer

Figure 6 shows that when a user selects an answer and presses the *'confirm'* button, the countdown freeze immediately and the button changes to *'next'*.

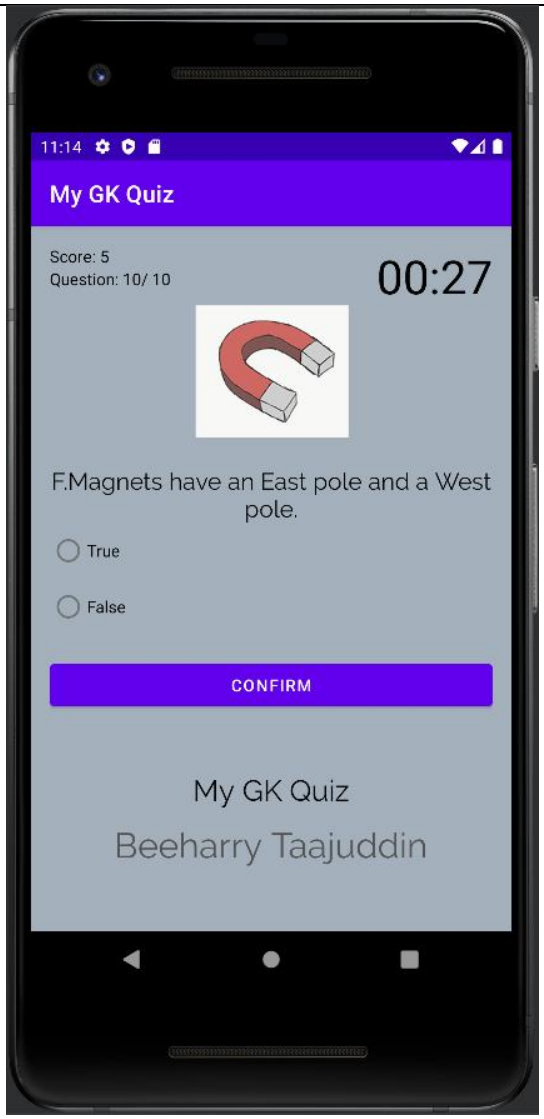


Figure 7 - Last Question

Figure 7 shows that the user has reached question 10 out of 10 with a score of 5. Figure 2 shows a high score of 4. The main page will now need to update the new top score automatically.

Figure 8 - Button Change

Figure 8 depicts the appearance of the button *'finish'* after the user has confirmed his answer to the last question.

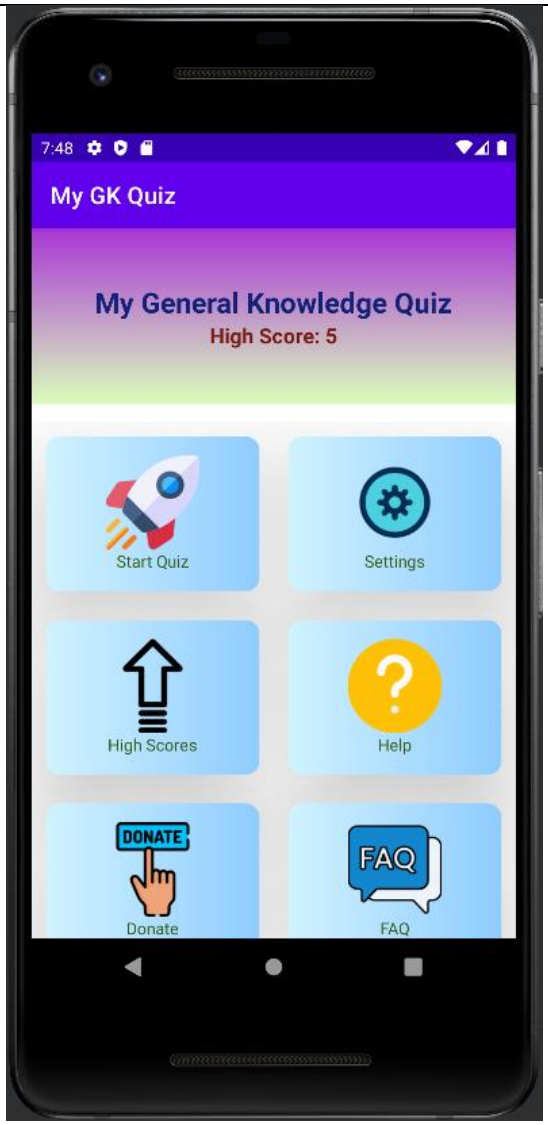


Figure 9 – High Score

Figure 9 depicts the main page's newly updated high score.

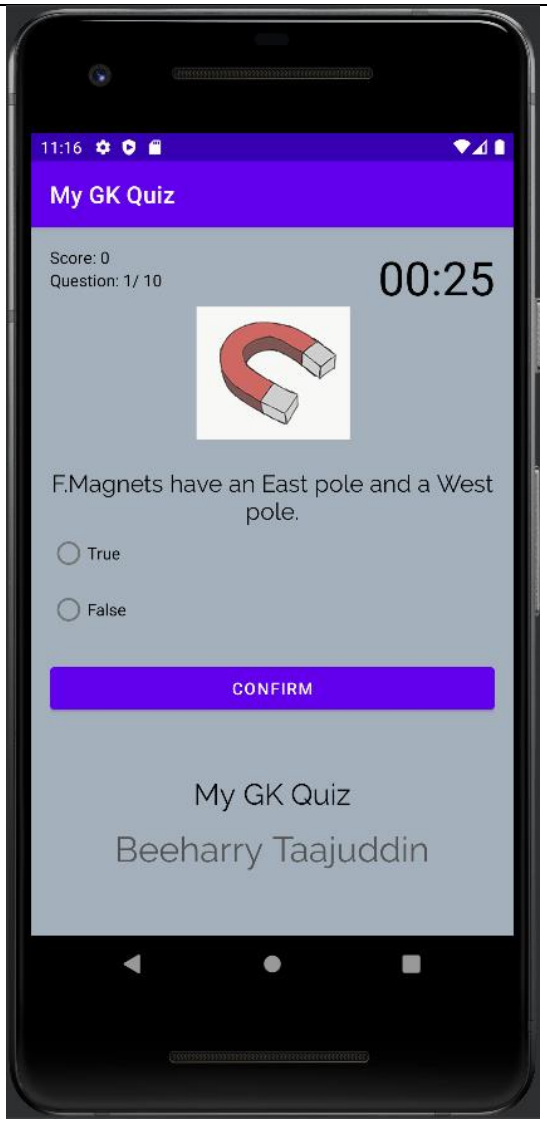


Figure 10 - Random Question

Picture 10 illustrates that the questions are displayed at random, as seen in figure 3 for question 1, it displays another question.

Implementation

Activity_splashscreen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    android:background="@drawable/gradient5"
    tools:context=".splashscreen">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Created by:"
        android:textColor="@color/black"
        android:textSize="20sp"
        android:fontFamily="@font/raleway"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="40dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Beeharry Taajuddin"
        android:fontFamily="@font/raleway"
        android:textColor="#545454"
        android:textSize="25sp"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="80dp"/>

    <ImageView
        android:layout_width="400dp"
        android:layout_height="400dp"
        android:src="@drawable/ic_resize"
        android:layout_centerInParent="true"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="My GK Quiz"
        android:textColor="@color/black"
        android:textSize="25sp"
        android:fontFamily="@font/raleway"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="110dp"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's test your IQ"
        android:fontFamily="@font/playlist"
        android:textColor="#545454"
        android:textSize="40sp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="40dp"/>

</RelativeLayout>
```

Splashscreen.java

```
12      @Override
13      protected void onCreate(Bundle savedInstanceState) {
14          super.onCreate(savedInstanceState);
15          setContentView(R.layout.activity_splashscreen);
16          getSupportActionBar().hide();
17          getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,WindowManager.LayoutParams.FLAG_FULLSCREEN);
18
19          new Handler().postDelayed(new Runnable() {
20              @Override
21              public void run() {
22                  Intent intent = new Intent( packageContext: splashscreen.this, MainActivity.class);
23                  startActivity(intent);
24                  finish();
25              }
26          }, delayMillis: 3000);
27      }
28  }
```

Figure 11 - Splash screen.java

In figure 11, **line 16** has been used to hide the *action bar* and **line 17** has been used to hide the *title bar*. For the splash screen to appear for 3 seconds, the code in **line 26** was used. And after the splash screen it redirects to the main activity.

Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView 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=".MainActivity">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="vertical">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="150dp"
            android:background="@drawable/gradient1"
            android:gravity="center"
            android:layout_marginBottom="15dp"
            android:orientation="vertical">

            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="My General Knowledge Quiz"
                android:textStyle="bold"
                android:textColor="#122279"
                android:textSize="24sp" />

            <TextView
                android:id="@+id/text_view_highscore"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:gravity="center"
                android:text="High Score: 0"
                android:textStyle="bold"
                android:textColor="#791e12"
                android:textSize="18sp" />
        </LinearLayout>

        <GridLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:alignmentMode="alignMargins"
            android:columnCount="2"
            android:columnOrderPreserved="false"
            android:rowCount="6">

            <androidx.cardview.widget.CardView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_rowWeight="1"
                android:layout_columnWeight="1"
                android:layout_margin="12dp"
                android:elevation="6dp"
                app:cardCornerRadius="12dp"
                app:cardElevation="60dp">

                <LinearLayout
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:gravity="center"
                    android:background="@drawable/gradient2"
                    android:orientation="vertical"
                    android:padding="16dp">

                    <ImageView
                        android:id="@+id/startQuizImg"
                        android:layout_width="80dp"
                        android:layout_height="80dp"
                        android:src="@drawable/start"></ImageView>

                    <TextView
                        android:layout_width="wrap_content"
                        android:textColor="#335c18"
                        android:layout_height="wrap_content"
                        android:text="Start Quiz"></TextView>
                </LinearLayout>
            </androidx.cardview.widget.CardView>

            <androidx.cardview.widget.CardView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_rowWeight="1"
                android:layout_columnWeight="1"
                android:layout_margin="12dp"
                android:elevation="6dp"
                app:cardCornerRadius="12dp"
                app:cardElevation="60dp">

                <LinearLayout
                    android:layout_width="match_parent"
                    android:background="@drawable/gradient2"
                    android:layout_height="match_parent"
                    android:gravity="center"
```

```
                android:orientation="vertical"
                android:padding="16dp">

                <ImageView
                    android:id="@+id/settingsImg"
                    android:layout_width="80dp"
                    android:layout_height="80dp"
                    android:src="@drawable/setup"></ImageView>

                <TextView
                    android:layout_width="wrap_content"
                    android:textColor="#335c18"
                    android:layout_height="wrap_content"
                    android:text="Settings"></TextView>
            </LinearLayout>
        </androidx.cardview.widget.CardView>

        <androidx.cardview.widget.CardView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_rowWeight="1"
            android:layout_columnWeight="1"
            android:layout_margin="12dp"
            android:elevation="6dp"
            app:cardCornerRadius="12dp"
            app:cardElevation="60dp">

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:background="@drawable/gradient2"
                android:gravity="center"
                android:orientation="vertical"
                android:padding="16dp">

                <ImageView
                    android:id="@+id/highScporesImg"
                    android:layout_width="80dp"
                    android:layout_height="80dp"
                    android:src="@drawable/upload"></ImageView>

                <TextView
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:textColor="#335c18"
                    android:text="High Scores"></TextView>
            </LinearLayout>
        </androidx.cardview.widget.CardView>

        <androidx.cardview.widget.CardView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_rowWeight="1"
            android:layout_columnWeight="1"
            android:layout_margin="12dp"
            android:elevation="6dp"
            app:cardCornerRadius="12dp"
            app:cardElevation="60dp">

            <LinearLayout
                android:background="@drawable/gradient2"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:gravity="center"
                android:orientation="vertical"
                android:padding="16dp">

                <ImageView
                    android:id="@+id/helpImg"
                    android:layout_width="80dp"
                    android:layout_height="80dp"
                    android:src="@drawable/question"></ImageView>

                <TextView
                    android:layout_width="wrap_content"
                    android:textColor="#335c18"
                    android:layout_height="wrap_content"
                    android:text="Help">

                </TextView>
            </LinearLayout>
        </androidx.cardview.widget.CardView>

        <androidx.cardview.widget.CardView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_rowWeight="1"
            android:layout_columnWeight="1"
            android:layout_margin="12dp"
            android:elevation="6dp"
            app:cardCornerRadius="12dp"
            app:cardElevation="60dp">

            <LinearLayout
                android:layout_width="match_parent"
```

```
                android:layout_height="match_parent"
                android:background="@drawable/gradient2"
                android:gravity="center"
                android:orientation="vertical"
                android:padding="16dp">

                <ImageView
                    android:id="@+id/donateImg"
                    android:layout_width="80dp"
                    android:layout_height="80dp"
                    android:src="@drawable/donate"></ImageView>

                <TextView
                    android:layout_width="wrap_content"
                    android:textColor="#335c18"
                    android:layout_height="wrap_content"
                    android:text="Donate">

                </TextView>
            </LinearLayout>
        </androidx.cardview.widget.CardView>

        <androidx.cardview.widget.CardView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_rowWeight="1"
            android:layout_columnWeight="1"
            android:layout_margin="12dp"
            android:elevation="6dp"
            app:cardCornerRadius="12dp"
            app:cardElevation="60dp">


            <LinearLayout
                android:layout_width="match_parent"
                android:background="@drawable/gradient2"
                android:layout_height="match_parent"
                android:gravity="center"
                android:orientation="vertical"
                android:padding="16dp">

                <ImageView
                    android:id="@+id/faqImg"
                    android:layout_width="80dp"
                    android:layout_height="80dp"
                    android:src="@drawable/faq"></ImageView>

                <TextView
                    android:layout_width="wrap_content"
                    android:textColor="#335c18"
                    android:layout_height="wrap_content"
                    android:text="FAQ">

                </TextView>
            </LinearLayout>
        </androidx.cardview.widget.CardView>
    </GridLayout>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:background="@drawable/gradient3"
        android:gravity="center"
        android:layout_marginTop="15dp"
        android:orientation="vertical">

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:text="BSE20AFT: Beeharry Taajuddin"
            android:textStyle="italic"
            android:textColor="#791e12"
            android:textSize="14sp" />
    </LinearLayout>
</LinearLayout>
</ScrollView>
```

MainActivity.java

```
15      private static final int REQUEST_CODE_QUIZ = 1;
16
17      public static final String SHARED_PREFS = "sharedPrefs";
18      public static final String KEY_HIGHSCORE = "keyHighscore";
19
20      private TextView textViewHighscore;
21
22      private int highscore;
23
24      ImageView startQuizImg;
```

Figure 12 - MainActivity.java (1)

In figure 12, variables, **SHARED_PREFS** and **KEY_HIGHSCORE** (that will be used as a key in share preference to save highscores) constant are created.

```
25      @Override
26      protected void onCreate(Bundle savedInstanceState) {
27          super.onCreate(savedInstanceState);
28          setContentView(R.layout.activity_main);
29
30          textViewHighscore = findViewById(R.id.text_view_highscore);
31          loadHighscore();
32
33          startQuizImg = findViewById(R.id.startQuizImg);
34          startQuizImg.setOnClickListener(new View.OnClickListener() {
35              @Override
36              public void onClick(View view) {
37                  startQuiz();
38              }
39          });
40      }
41
42      private void startQuiz() {
43          Intent intent = new Intent( packageContext: MainActivity.this, PlayGameActivity.class);
44          startActivityForResult(intent, REQUEST_CODE_QUIZ);
45      }
```

Figure 13 - MainActivity.java (2)

In figure 13, at *line 36*, an **onclick** method is used to make the card clickable. An *activity PlayGameActivity* has been created so that intent has been used to open the *PlayGameActivity*.

```
42      private void startQuiz() {
43          Intent intent = new Intent( packageContext: MainActivity.this, PlayGameActivity.class);
44          startActivityForResult(intent, REQUEST_CODE_QUIZ);
45      }
46
47      @Override
48      protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
49          super.onActivityResult(requestCode, resultCode, data);
50
51          if (requestCode == REQUEST_CODE_QUIZ) {
52              if (resultCode == RESULT_OK) {
53                  int score = data.getIntExtra(PlayGameActivity.EXTRA_SCORE,  defaultValue: 0);
54                  if (score > highscore) {
55                      updateHighscore(score);
56                  }
57              }
58          }
59      }
```

Figure 14 - MainActivity.java (3)

In figure 14, at line 42, **startQuiz()** method is created. Then to get user's score, **startActivityForResult** is used. An intent and a request code are passed to **startActivityResult** so that the result may be traced back to its source and for this purpose a constant called **REQUEST_CODE_QUIZ** is defined.

The **requestCode** is passed by the method **onActivityResult** which is compared to the **REQUEST_CODE_QUIZ** generated since this is the result that will be handled. An if statement is utilized on *line 51*, to check whether the **requestCode** equals **REQUEST_CODE_QUIZ**. The second if statement is utilized to check if the **resultCode** is equal to **RESULT_OK**. The new variable is saved in a new integer called **score** and **data.getInExtra** is utilized to past the name for which EXTRA_SCORE constant is defined in the PlayGameActivity.

The last if statement (*line 54*) is utilized to check if score is more than high score. If this is the case, **updateHighscore(score)** is called.

```
61      private void loadHighscore() {
62          SharedPreferences prefs = getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
63          highscore = prefs.getInt(KEY_HIGHSCORE,  defValue: 0);
64          textViewHighscore.setText("High Score: " + highscore);
65      }
66
67      private void updateHighscore(int highscoreNew) {
68          highscore = highscoreNew;
69          textViewHighscore.setText("High Score: " + highscore);
70
71          SharedPreferences prefs = getSharedPreferences(SHARED_PREFS, MODE_PRIVATE);
72          SharedPreferences.Editor editor = prefs.edit();
73          editor.putInt(KEY_HIGHSCORE, highscore);
74          editor.apply();
75      }
76  }
```

Figure 15 - MainActivity.java (4)

Figure 15 shows that, at line 61, the **loadHighscore()** is define because when the app is launched, the high score must be loaded.

The high score is set to the new score at *line 67* f the method **updateHighscore**. The new high score is kept in the **SharedPreferences** and the **textViewHighscore** is set to it. At *line 73*, **editor.putInt** is utilized to save the value of high score by passing the **KEY_HIGHSCORE** and **highscore**. At last, the data is saved using the **editor.apply()**.

Activity_play_game.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    android:background="#a4b1bc"
    android:padding="16dp"
    tools:context=".PlayGameActivity">

    <TextView
        android:id="@+id/text_view_score"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Score: 0"
        android:textColor="@color/black" />

    <TextView
        android:id="@+id/text_view_question_count"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Questions: 1/x"
        android:textColor="@color/black"
        android:layout_below="@id/text_view_score"/>

    <TextView
        android:id="@+id/text_view_countdown"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:textColor="@color/black"
        android:text="00:30"
        android:layout_alignParentEnd="true"/>

    <ImageView
        android:id="@+id/image_view"
        android:layout_width="131dp"
        android:layout_height="117dp"
        android:layout_gravity="center"
        android:layout_marginLeft="125dp"
        android:layout_marginTop="50dp"
        android:scaleType="fitCenter"
        android:src="@drawable/img_0512" />

    <TextView
        android:id="@+id/text_view_question"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/radio_group"
        android:fontFamily="@font/raleway"
        android:text="Question\nQuestion\nQuestion"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="20sp" />

    <RadioGroup
        android:id="@+id/radio_group"
        android:layout_centerVertical="true"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">

        <RadioButton
            android:id="@+id/radio_button1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="True" />

        <RadioButton
            android:id="@+id/radio_button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="False" />

    </RadioGroup>

    <Button
        android:id="@+id/button_confirm_next"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/radio_group"
        android:layout_marginTop="19dp"
        android:text="Confirm" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="90dp"
        android:fontFamily="@font/raleway"
        android:text="My GK Quiz"
        android:textColor="@color/black"
        android:textSize="25sp" />

    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Beeharry Taajuddin"
        android:fontFamily="@font/raleway"
        android:textColor="#545454"
        android:textSize="30sp"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        android:layout_marginBottom="40dp"/>

</RelativeLayout>
```

PlayGameActivity.java

```
23      public static final String EXTRA_SCORE = "extraScore";
24      private static final long COUNTDOWN_IN_MILLIS = 30000;
25
26      private ImageView mImageView;
27      private TextView textViewQuestion;
28      private TextView textViewScore;
29      private TextView textViewQuestionCount;
30      private TextView textViewCountDown;
31      private RadioGroup rbGroup;
32      private RadioButton rb1;
33      private RadioButton rb2;
34      private Button buttonConfirmNext;
35
36      private ColorStateList textColorDefaultRb;
37      private ColorStateList textColorDefaultCd;
38
39      private CountDownTimer countDownTimer;
40      private long timeLeftInMillis;
41
42      private List<Question> questionList;
43      private int questionCounter;
44      private int questionCountTotal;
45      private Question currentQuestion;
46
47      private int score;
48      private boolean answered;
```

Figure 16 - PlayGameActivity.java (1)

In figure 16, all the variable is created first and in **line 24**, the timer has been set to 30 seconds. For lines **23** and **24**, a constant **EXTRA_SCORE** is defined and set to **extraScore** to transmit the score value from one activity to another.

```
51      @Override
52      protected void onCreate(Bundle savedInstanceState) {
53          super.onCreate(savedInstanceState);
54          setContentView(R.layout.activity_play_game);
55
56          mImageView = findViewById(R.id.image_view);
57          textViewQuestion = findViewById(R.id.text_view_question);
58          textViewScore = findViewById(R.id.text_view_score);
59          textViewQuestionCount = findViewById(R.id.text_view_question_count);
60          textViewCountDown = findViewById(R.id.text_view_countdown);
61          rbGroup = findViewById(R.id.radio_group);
62          rb1 = findViewById(R.id.radio_button1);
63          rb2 = findViewById(R.id.radio_button2);
64          buttonConfirmNext = findViewById(R.id.button_confirm_next);
65
66          textColorDefaultRb = rb1.getTextColors();
67          textColorDefaultCd = textViewCountDown.getTextColors();
```

Figure 17 - PlayGameActivity.java (2)

Figure 17 depicts that all variables are assigned to their respective id in the xml layout in the ***onCreate*** method.

```
69          QuizDbHelper dbHelper = new QuizDbHelper( context: this);
70          questionList = dbHelper.getAllQuestions();
71          questionCountTotal = questionList.size();
72          Collections.shuffle(questionList);
73
74          showNextQuestion();
75
76          buttonConfirmNext.setOnClickListener(new View.OnClickListener() {
77              @Override
78              public void onClick(View v) {
79                  if (!answered) {
80                      if (rb1.isChecked() || rb2.isChecked()) {
81                          checkAnswer();
82                      } else {
83                          Toast.makeText( context: PlayGameActivity.this,  text: "Please select an answer", Toast.LENGTH_SHORT).show();
84                      }
85                  } else {
86                      showNextQuestion();
87                  }
88              }
89          });
```

Figure 18 - PlayGameActivity.java (3)

In figure 18, the **QuizDbHelper** is initialized at *line 69*. The method **dbHelper** is then used to populate the **questionList** with data **.getAllQuestions()**. When the method is invoked, it also creates the database for the first time Now to show the question at random, *line 72*, **Collections.shuffle(questionList)** is used and the **questionList** is passed into. Moreover, a separated method **showNextQuestion()** is used to called the next question.

An *onclick listener* is set on the **buttonConfirmNext**. Two if statements have been used. For the first if statement, it checks whether the answer is answered. If yes, the method **showNextQuestion()** is called. The second if statement checks whether the radio button true false is selected or not and an **isChecked()** is used for that. If it is selected, **checkAnswer()** method is called else a **Toast** is used to display message.

```
92      private void showNextQuestion() {
93          rb1.setTextColor(textColorDefaultRb);
94          rb2.setTextColor(textColorDefaultRb);
95          rbGroup.clearCheck();
96
97          if (questionCounter < 10) {
98              currentQuestion = questionList.get(questionCounter);
99
100             mImageView.setImageResource(currentQuestion.getImage());
101             textViewQuestion.setText(currentQuestion.getQuestion());
102             rb1.setText(currentQuestion.getOption1());
103             rb2.setText(currentQuestion.getOption2());
104
105             questionCounter++;
106             textViewQuestionCount.setText("Question: " + questionCounter + "/ 10");
107             answered = false;
108             buttonConfirmNext.setText("Confirm");
109
110             timeLeftInMillis = COUNTDOWN_IN_MILLIS;
111             startCountDown();
112         } else {
113             finishQuiz();
114         }
115     }
```

Figure 19 - PlayGameActivity.java (4)

A private **showNextQuestion()** method is developed as shown in figure 19. The **rbGroup.clearCheck()** is used to clear the radio buttons when another question is displayed. *Line 97*, an if statement has been used to check if **questionCounter < 10.** As per specifications, the user must have a 10 questions in a large set of 20 when the quiz game starts every time. In case, the **questionCounter** reached 10, the quiz will end since **finishQuiz()** method is used at *line 113*. Else the following question is saved in the variable **currentQuestion** (*line 98*).

Line 100, is the variable **mImageView** which has been used to display the next question and **.setText** is used to passed the **currentQuestion** and it is the same for the *line 101* to *103*.

At *line 105*, **questionCounter++** is used to increment the question counter by 1 and **textViewQuestionCount** variable is used to display the amount of question left by passing the **questionCounter**.

At *line 107*, the answer is set to false since the answer will be locked (***answered = false***) if the user clicks the button confirm rather than going onto the following question. Then **.setText("confirm")** is used to change the text button to next and if the answer is not locked, the text button will remain confirm.

```
117     private void startCountDown() {
118         countDownTimer = new CountDownTimer(timeLeftInMillis,  countDownInterval: 1000) {
119             @Override
120             public void onTick(long millisUntilFinished) {
121                 timeLeftInMillis = millisUntilFinished;
122                 updateCountDownText();
123             }
124
125             @Override
126             public void onFinish() {
127                 timeLeftInMillis = 0;
128                 updateCountDownText();
129                 checkAnswer();
130             }
131         }.start();
132     }
133
134     private  void updateCountDownText() {
135         int minutes = (int) (timeLeftInMillis / 1000) / 60;
136         int seconds = (int) (timeLeftInMillis / 1000) % 60;
137
138         String timeFormatted = String.format(Locale.getDefault(),  format: "%02d:%02d", minutes, seconds);
139
140         textViewCountDown.setText(timeFormatted);
141
142         if (timeLeftInMillis < 10000) {
143             textViewCountDown.setTextColor(Color.RED);
144         } else {
145             textViewCountDown.setTextColor(textColorDefaultCd);
146         }
147     }
```

Figure 20 - PlayGameActivity.java (5)

In figure 20, a **private void startCountDown()** has been implemented for the countdown. To do this, the **CountDownTimer** class is utilized to pass a running time in milliseconds along with an interval of how frequently it's on tick method is called and the *TextView* is then refreshed for every tick.

Lines 135 and ***136*** convert milliseconds to minutes and seconds which are then formatted into a string using the **String.format** method. Furthermore, after the countdown reaches less than 10 seconds, the **.setTextColor** method is called to change the text color of the countdown and when the timer expires, the **onFinish** method is invoked to check for the response.

```
149     private void checkAnswer() {
150         answered = true;
151
152         countDownTimer.cancel();
153
154         RadioButton rbSelected = findViewById(rbGroup.getCheckedRadioButtonId());
155         int answerNr = rbGroup.indexOfChild(rbSelected) + 1;
156
157         if (answerNr == currentQuestion.getAnswerNr()) {
158             score++;
159             textViewScore.setText("Score: " + score);
160         }
161
162         showSolution();
163     }
```

Figure 21 - PlayGameActivity.java (6)

Now to check the answer, in figure 21, the method **checkAnswer()** is defined. *Line 150*, because the question was answered, the answered is set to true. **rbSelected** variable is created and it stores the id of the radio button that is selected because the correct answer is saved in the database table as number, the selected button is converted to a number by creating an **int answerNr** in *line 155*. Then **rbSelected** is passed by **rbGroup.indexOfChild(rbSelected)** and this returns an index of the radio button in the radio group that was picked.

Furthermore, the **+1** is added, because the index begins at 0 but the **answerNr** begins at 1. When one of the radio buttons is selected, **answerNr** variable is set to 1 and an if statement is used to determine whether the selected answerNr is the same as the proper **answerNr** for that question and if so, the question was correctly answered.

Line 158, the score is incremented by 1 and the t**extViewScore** is updated to reflect the new score.

At last, whether the response is right or wrong, the **showSolution()** method is called.

```
165     private void showSolution() {
166         rb1.setTextColor(Color.RED);
167         rb2.setTextColor(Color.RED);
168
169         switch (currentQuestion.getAnswerNr()) {
170             case 1:
171                 rb1.setTextColor(Color.GREEN);
172                 textViewQuestion.setText("Answer True is correct");
173                 break;
174             case 2:
175                 rb2.setTextColor(Color.GREEN);
176                 textViewQuestion.setText("Answer False is correct");
177                 break;
178         }
179
180         if (questionCounter < 10) {
181             buttonConfirmNext.setText("Next");
182         } else {
183             buttonConfirmNext.setText("Finish");
184         }
185     }
```

Figure 22 - PlayGameActivity.java (7)

In figure 22, the method **checkAnswer()** is defined. Initially, the radio buttons **rb1** and **rb2** are set to *red*. To determine which answer (***answerNr***) is correct, a *switch* statement is used, and the radio button (either *rb1* or *rb2*) is set to green and to show which response is right, the **textViewQuestion** is utilized.

An if statement is implemented to check if **questionCounter** is less than 10. In case there are more questions, the button (***buttonConfirmNext***) would be **Next** else **Finish** which will end the quiz game.

Figure 23 - PlayGameActivity.java (8)

In figure 23, **finishQuiz()** is created to end the game. Before that, the score in send from one activity to the next. The **putExtra** is called by the **resultIntent** as it requires a name and a value to be passed into it. For the *name*, **EXTRA_SCORE** constant is used and for the *value*, **score** is used.

Afterwards, **setResult** is called with **RESULT_OK**, **resultIntent** is passed for the result value and to close the activity **finish()** is used (*line 191*).

Line 195 **onDestroy()** is created to cancel the **countDownTimer**.

Question.java

```
3   public class Question {
4       private int image;
5       private String question;
6       private String option1;
7       private String option2;
8       private int answerNr;
9
10      public Question() {}
11
12      public Question( int image, String question, String option1, String option2, int answerNr) {
13          this.image = image;
14          this.question = question;
15          this.option1 = option1;
16          this.option2 = option2;
17          this.answerNr = answerNr;
18      }
```

Figure 24 - Question.java (1)

The Question.java class was designed to store information for questions with various answer options, correct answers and to act as a bridge between the application's SQLite database as shown in 24. Form ***line 12*** to ***17***, to pass all of these five values and set them to the instance variables, a constructor is developed.

```
22      public int getImage() { return image; }
25
26      public void setImage(int image) { this.image = image; }
29
30      public String getQuestion() { return question; }
33
34      public void setQuestion(String question) { this.question = question; }
37
38      public String getOption1() { return option1; }
41
42      public void setOption1(String option1) { this.option1 = option1; }
45
46      public String getOption2() { return option2; }
49
50      public void setOption2(String option2) { this.option2 = option2; }
53
54      public int getAnswerNr() { return answerNr; }
57
58      public void setAnswerNr(int answerNr) { this.answerNr = answerNr; }
61  }
```

Figure 25 - Question.java (2)

To alter the fields in the question class and to obtain all the values out of these classes to display them, **setter** and **getter** techniques are utilized as shown in figure 25 above.

QuizContract.java

```
5  public final class QuizContract {
6
7      private QuizContract() {
8
9      }
10
11     public static class QuestionsTable implements BaseColumns {
12         public  static final String TABLE_NAME = "quiz_questions";
13         public  static final String COLUMN_IMAGE = "image";
14         public  static final String COLUMN_QUESTION = "question";
15         public  static final String COLUMN_OPTION1 = "option1";
16         public  static final String COLUMN_OPTION2 = "option2";
17         public  static final String COLUMN_ANSWER_NR = "answer_nr";
18
19     }
20 }
```

Figure 26 - QuizContract.java

Figure 26 shows the **QuizContract.java** class which is nothing more than a container for various constants used in SQLite database. In case, if we need to alter the name of a column, we can do so in one place rather than having to update it across our code. Apart from giving these constants, this class will do nothing else.

Table in the database has its own inner class. Constant are defined which will include the same information as *question.java* and the table's name. They're string values that are public so that they can be accessed outside of this class and so that the values won't change. There will be only one table in the database: ***quiz_questions***.

As the **BaseColumns** provides an interface for the ***_id***, it is utilized. This constant *_id* will be utilized to build a new database column. If more questions are added, the *_id* will automatically increment in the tables.

QuizDbHelper.java

```
18      private static final String DATABASE_NAME = "MyGKQuiz.db";
19      private static final int DATABASE_VERSION = 1;
20
21      private SQLiteDatabase db;
22
23
24      public QuizDbHelper(Context context) { super(context, DATABASE_NAME, factory: null, DATABASE_VERSION); }
27
28      @Override
29  @   public void onCreate(SQLiteDatabase db) {
30          this.db = db;
31
32          final String SQL_CREATE_QUESTIONS_TABLE = "CREATE TABLE " +
33                  QuestionsTable.TABLE_NAME + " ( " + QuestionsTable._ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
34                  QuestionsTable.COLUMN_IMAGE + " INTEGER, " +
35                  QuestionsTable.COLUMN_QUESTION + " TEXT, " +
36                  QuestionsTable.COLUMN_OPTION1 + " TEXT, " +
37                  QuestionsTable.COLUMN_OPTION2 + " TEXT, " +
38                  QuestionsTable.COLUMN_ANSWER_NR + " INTEGER" + ")";
39
40
41          db.execSQL(SQL_CREATE_QUESTIONS_TABLE);
42          fillQuestionsTable();
43      }
```

Figure 27 - QuizDbHelper.java (1)

In figure 27, *line 18* and *19*, the **DATABASE_NAME** and **DATABASE_VERSION** variables are first created. The variable **SQLiteDatabase db** is used to store the database's reference.

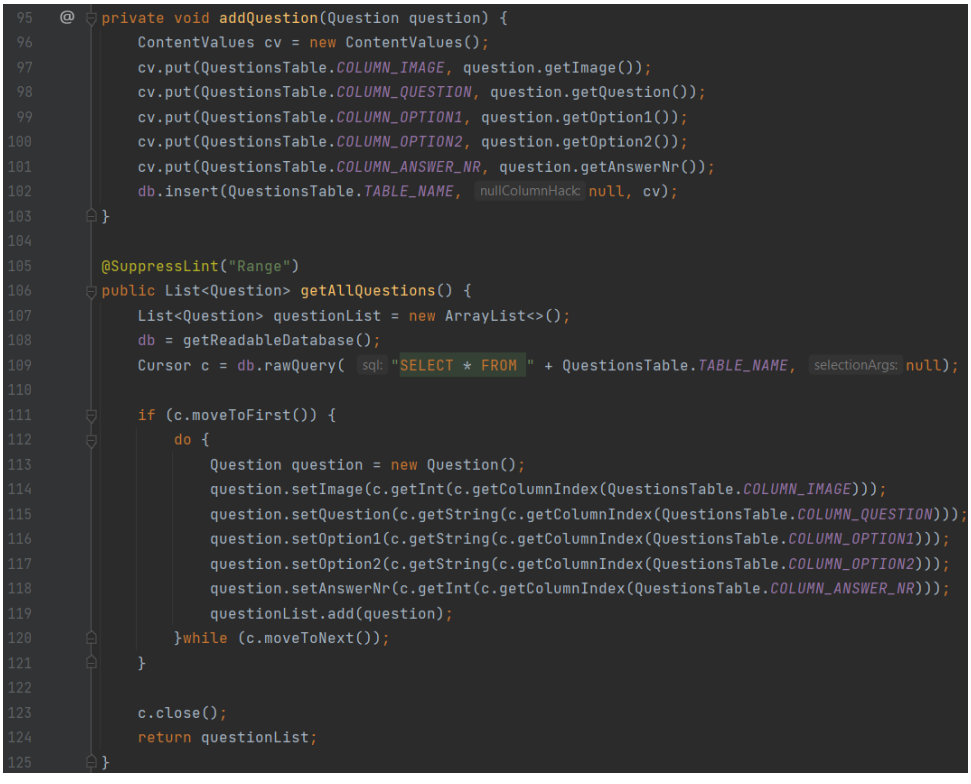
When accessing the database for the first time, the **onCreate** method is called and inside the method **this.db = db**, the first database is created which is utilized as a reference outside the **onCreate**. The SQLITE command is used to construct the database. By passing the string **SQL_CREATE_QUESTIONS_TABLE** to **db.execSQL**, the SQLite command is executed. The **fillQuestionsTable()** method is used to populate it with questions after the database has been created.

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    db.execSQL("DROP TABLE IF EXISTS " + QuestionsTable.TABLE_NAME);
    // create fresh books table
    onCreate(db);
}

private void fillQuestionsTable() {
    Question q1 = new Question(R.drawable.img_0512, question: "T.Magnets have two poles.", option1: "True", option2: "False", answerNr: 1);
    addQuestion(q1);
    Question q2 = new Question(R.drawable.img_0516, question: "F.Magnets have an East pole and a West pole.", option1: "True", option2: "False",
    addQuestion(q2);
    Question q3 = new Question(R.drawable.img_0521, question: "F.The earth is not magnetic.", option1: "True", option2: "False", answerNr: 2);
    addQuestion(q3);
    Question q4 = new Question(R.drawable.img_0522, question: "F.Magnetic fields can clearly be seen with your eyes.", option1: "True", option
    addQuestion(q4);
    Question q5 = new Question(R.drawable.img_0523, question: "T.The above diagram represents attraction between two magnets.", option1: "Tru
    addQuestion(q5);
    Question q6 = new Question(R.drawable.img_0525, question: "T.An iron nail is attracted to a magnet.", option1: "True", option2: "False", a
    addQuestion(q6);
    Question q7 = new Question(R.drawable.img_0526, question: "F.An electromagnet uses alternating current from the mains.", option1: "True",
    addQuestion(q7);
    Question q8 = new Question(R.drawable.img_0527, question: "T.An electromagnet is a temporary magnet.", option1: "True", option2: "False", a
    addQuestion(q8);
    Question q9 = new Question(R.drawable.img_0528, question: "T.A magnet is a disc magnet.", option1: "True", option2: "False", answerNr: 1);
    addQuestion(q9);
    Question q10 = new Question(R.drawable.img_0529, question: "T.When an unbalanced force acts on an object, the force causes it to acceler
    addQuestion(q10);
    Question q11 = new Question(R.drawable.img_0530, question: "T.Matter is pulled to the ground by gravity.", option1: "True", option2: "False'
    addQuestion(q11);
    Question q12 = new Question(R.drawable.img_0531, question: "F.Does 'Plasma' is the process of vaporization that occurs throughout a liqu
    addQuestion(q12);
    Question q13 = new Question(R.drawable.img_0532, question: "F.The change from the liquid to the gas state of matter is known as Melting.
```

Figure 28 - QuizDbHelper.java (2)

Figure 28 shows, if ever there is an update, the method **onCreate** will not be invoked. The procedure for adding questions is depicted in *line 52*. A constructor is used to deliver the question, the different answers, and the answer number to their respective variables, for example *q1*. Then variable ***q1*** is passed into the **addQuestion()** method in order for the question to be added.



```
private void addQuestion(Question question) {
    ContentValues cv = new ContentValues();
    cv.put(QuestionsTable.COLUMN_IMAGE, question.getImage());
    cv.put(QuestionsTable.COLUMN_QUESTION, question.getQuestion());
    cv.put(QuestionsTable.COLUMN_OPTION1, question.getOption1());
    cv.put(QuestionsTable.COLUMN_OPTION2, question.getOption2());
    cv.put(QuestionsTable.COLUMN_ANSWER_NR, question.getAnswerNr());
    db.insert(QuestionsTable.TABLE_NAME,  nullColumnHack: null, cv);
}

@SuppressLint("Range")
public List<Question> getAllQuestions() {
    List<Question> questionList = new ArrayList<>();
    db = getReadableDatabase();
    Cursor c = db.rawQuery( sql: "SELECT * FROM " + QuestionsTable.TABLE_NAME,  selectionArgs: null);

    if (c.moveToFirst()) {
        do {
            Question question = new Question();
            question.setImage(c.getInt(c.getColumnIndex(QuestionsTable.COLUMN_IMAGE)));
            question.setQuestion(c.getString(c.getColumnIndex(QuestionsTable.COLUMN_QUESTION)));
            question.setOption1(c.getString(c.getColumnIndex(QuestionsTable.COLUMN_OPTION1)));
            question.setOption2(c.getString(c.getColumnIndex(QuestionsTable.COLUMN_OPTION2)));
            question.setAnswerNr(c.getInt(c.getColumnIndex(QuestionsTable.COLUMN_ANSWER_NR)));
            questionList.add(question);
        }while (c.moveToNext());
    }

    c.close();
    return questionList;
}
```

Figure 29 - QuizDbHelper.java (3)

In figure 29, the question is inserted into the database using the **addQuestion(Question question)** method. The **ContentValues** class called cv is defined and to pass the key which is the column name as well as the values (***question.getQuestion()***) is passed when the user wants to input, **cv.put** is utilized. At *line 102*, the **db** variable which is saved inside the *onCreate* method, is utilized to put these values into the database and the **.insert** is called, table name and *cv* content values are passed to **db.insert(QuestionsTable.TABLE_NAME, null,cv)**.

The questions are retrieved from the database using the approach described at *line 106*. It's public since it'll be called from another class later to receive all the questions. To retrieve the data, an array list called **questionList** is constructed and **db = getReadableDatabase()** is used as a database reference. The database query is passed to **cursor c** as a string **("SELECT * FROM " + QuestionsTable.TABLE_NAME, null)** for the database query.

When the first entry exists, **c.moveToFirst()** is used to move the cursor to it, and when it does, the query within the do while loop is done, filling the question object with its respective contents. Only if the following element exists, **c.moveToNext()** is used to move the pointer to it.

Testing

Main menu

Description: The entry page of the application should consist of a main menu.

Result:

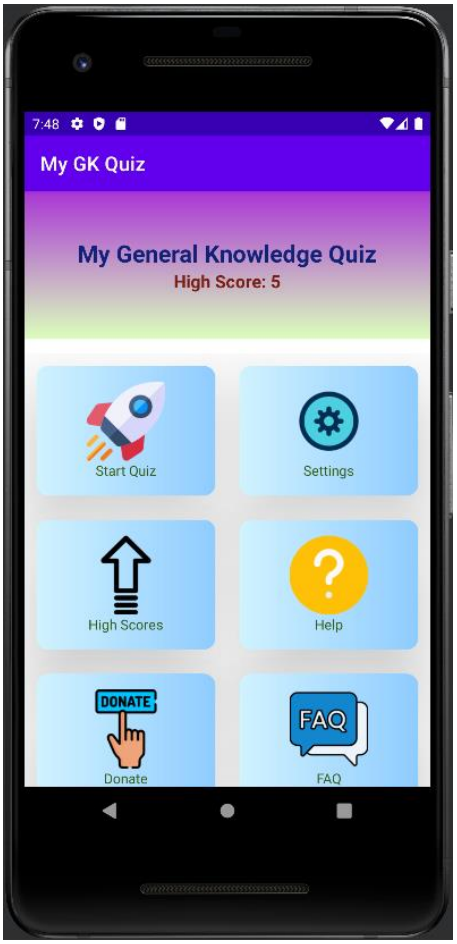


Figure 30 - Testing Main menu

Status: Pass.
Post-Conditions: The entry page of the application consist of a main menu.

Play quiz

Description: User should be able to play quiz.

Result:

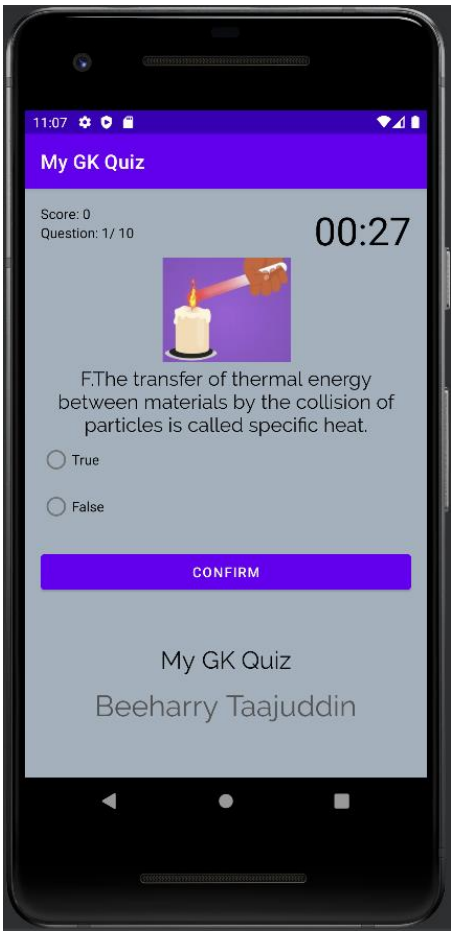


Figure 31 - Testing Play quiz

Status: Pass.
Post-Conditions: User is successfully able to play the quiz.

Countdown

Description: The countdown should begin as soon as the game begins.

Result:

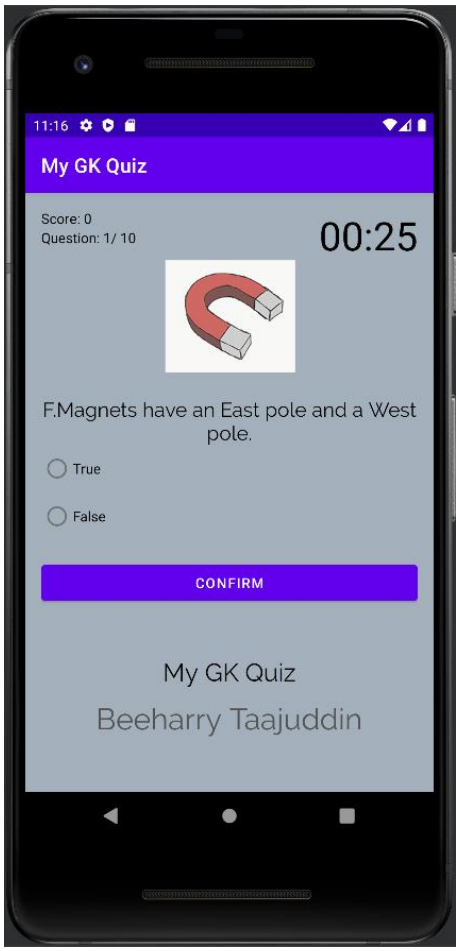


Figure 32 - Testing Countdown

Status: Pass.
Post-Conditions: The countdown has begun successfully.

Display 10 questions only

Description: Each quiz game should display only 10 questions.

Result:

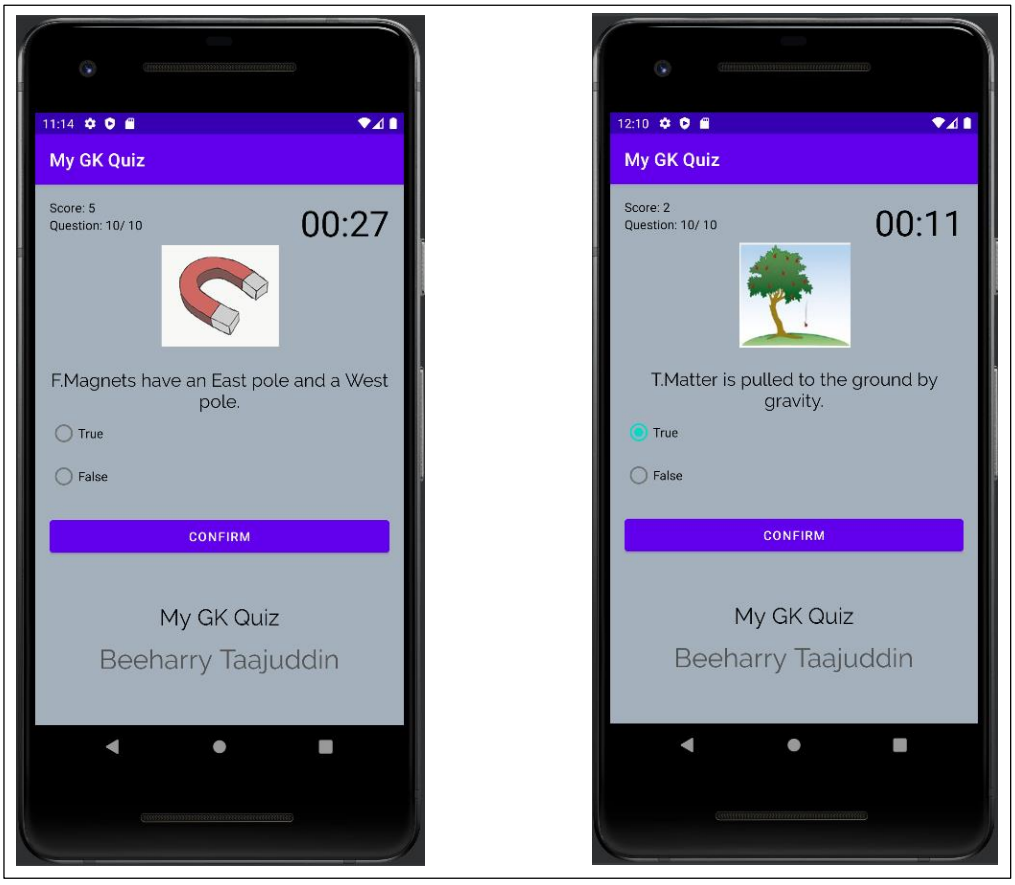


Figure 33 - Testing Display 10 question only

Status: Pass.
Post-Conditions: Only 10 questions display successfully.

Display score

Description: The user should be able to get (1) one mark for a correct answer and (0) zero for an incorrect answer.

Result:

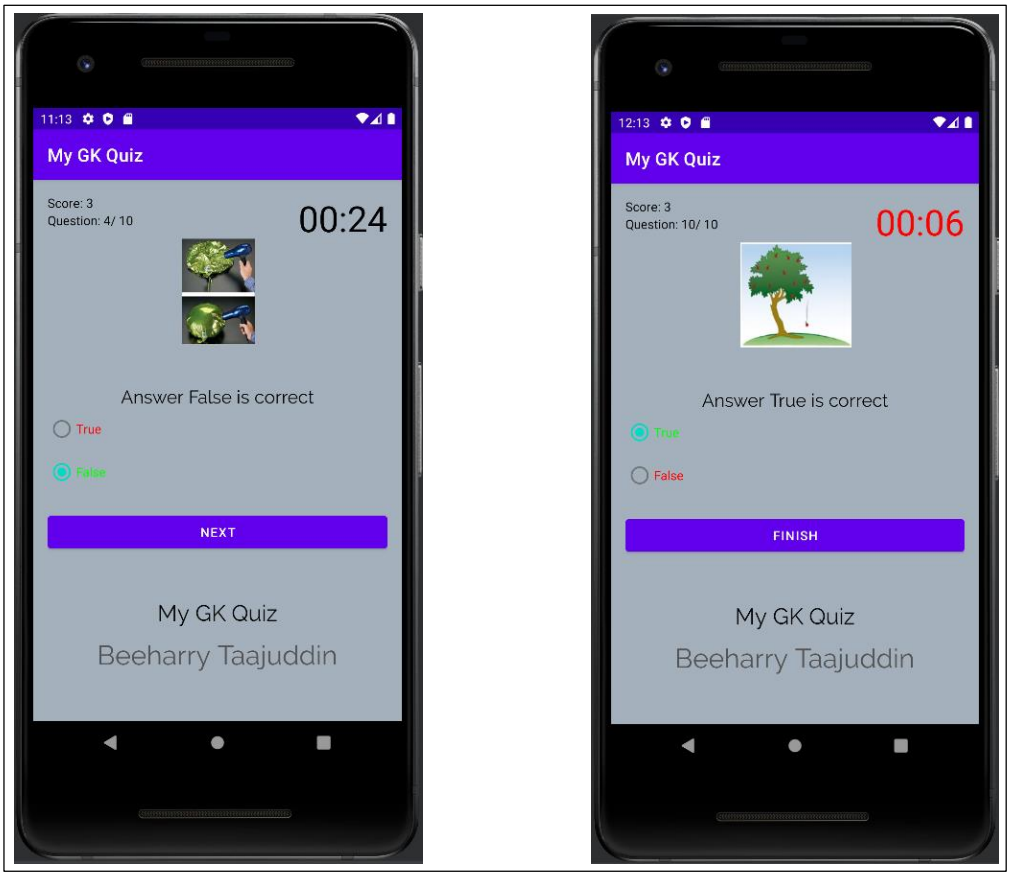


Figure 34 – Testing Display score

Status: Pass.
Post-Conditions: The user gets (1) one mark for a correct answer and (0) zero for an incorrect answer.

Display random question

Description: The GK Quiz app should be able to display questions randomly.

Result:

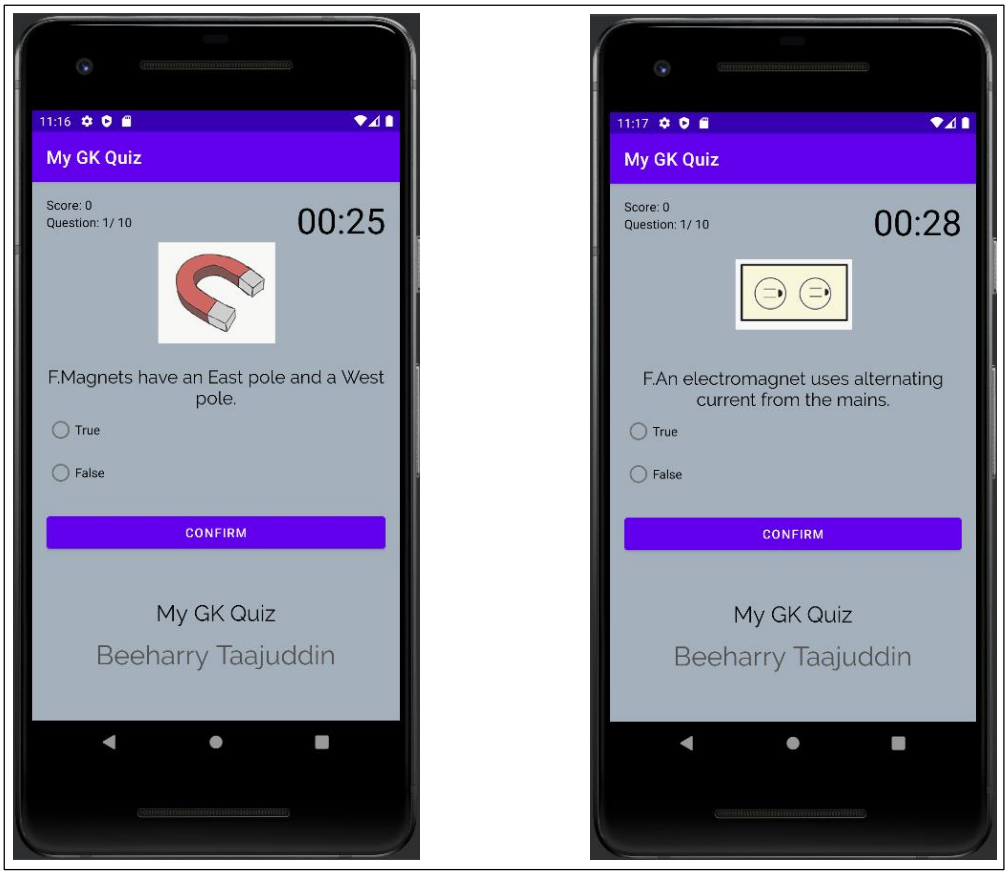


Figure 35 - Testing Display random question

Status: Pass.
Post-Conditions: Random questions displayed successfully.

Display correct answer

Description: The GK Quiz app should be able to display the correct answer for each question.

Result:

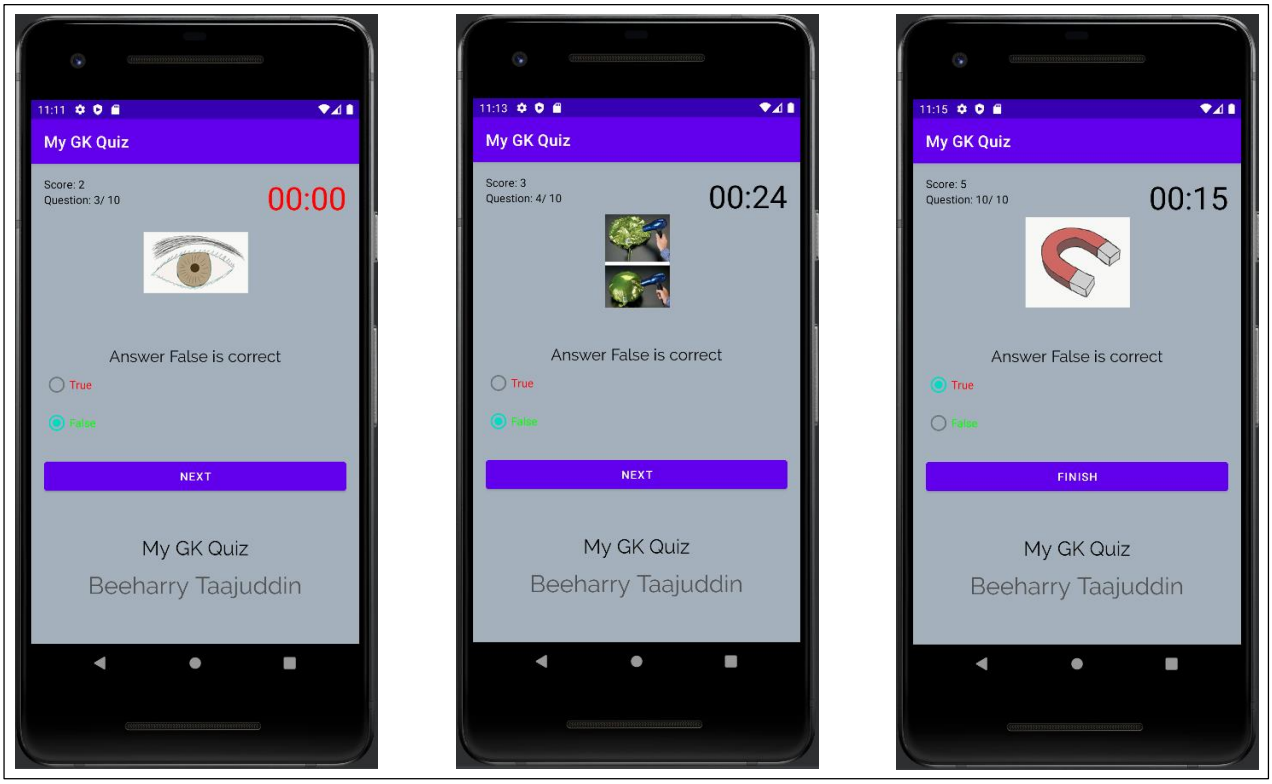


Figure 36 - Testing Display correct answer

Status: Pass.
Post-Conditions: Correct answer is displayed successfully.

View question's score & number of questions

Description: User should be able to view his score and number of question out of 10.

Result:

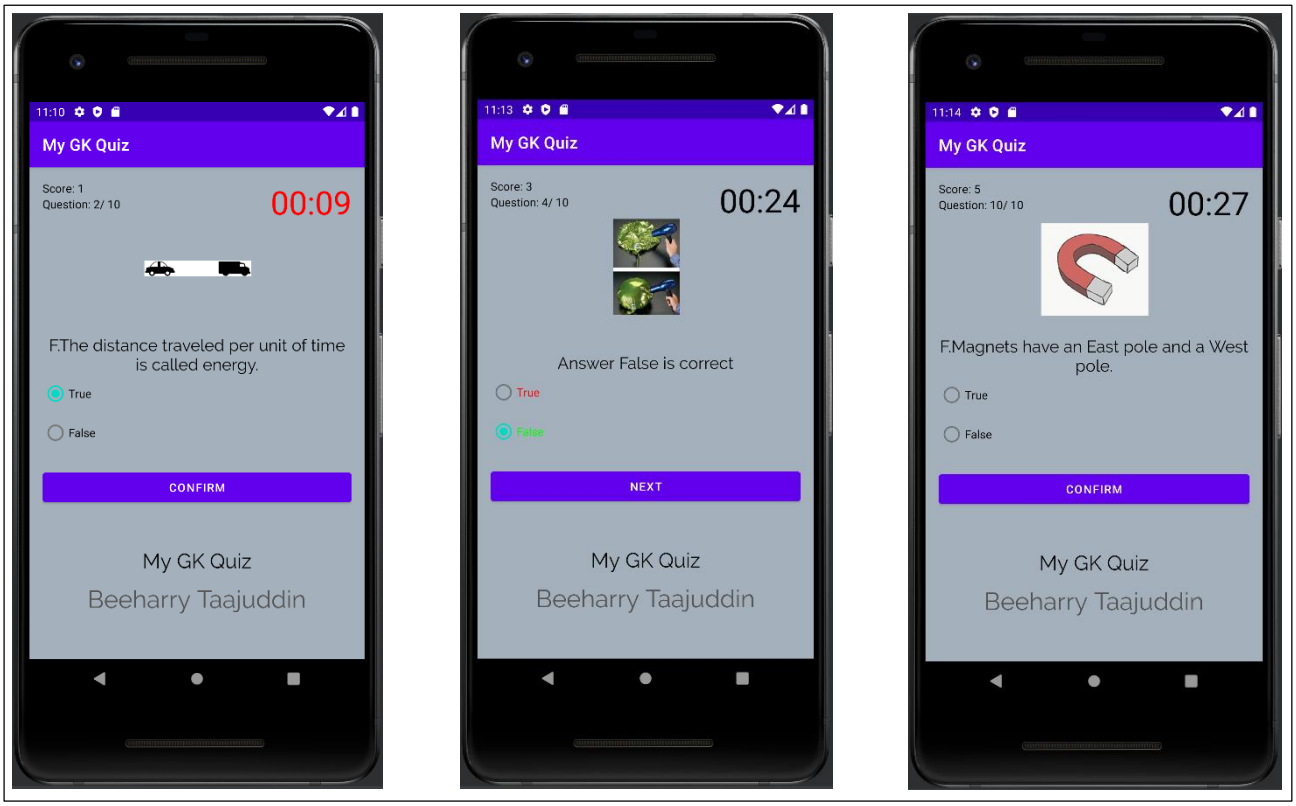


Figure 37 - Testing View question's score & number of questions

Status: Pass.

Post-Conditions: User can view his score and the number of questions attempted out of 10.

View high score

Description: User should be able to view his highest score.

Result:

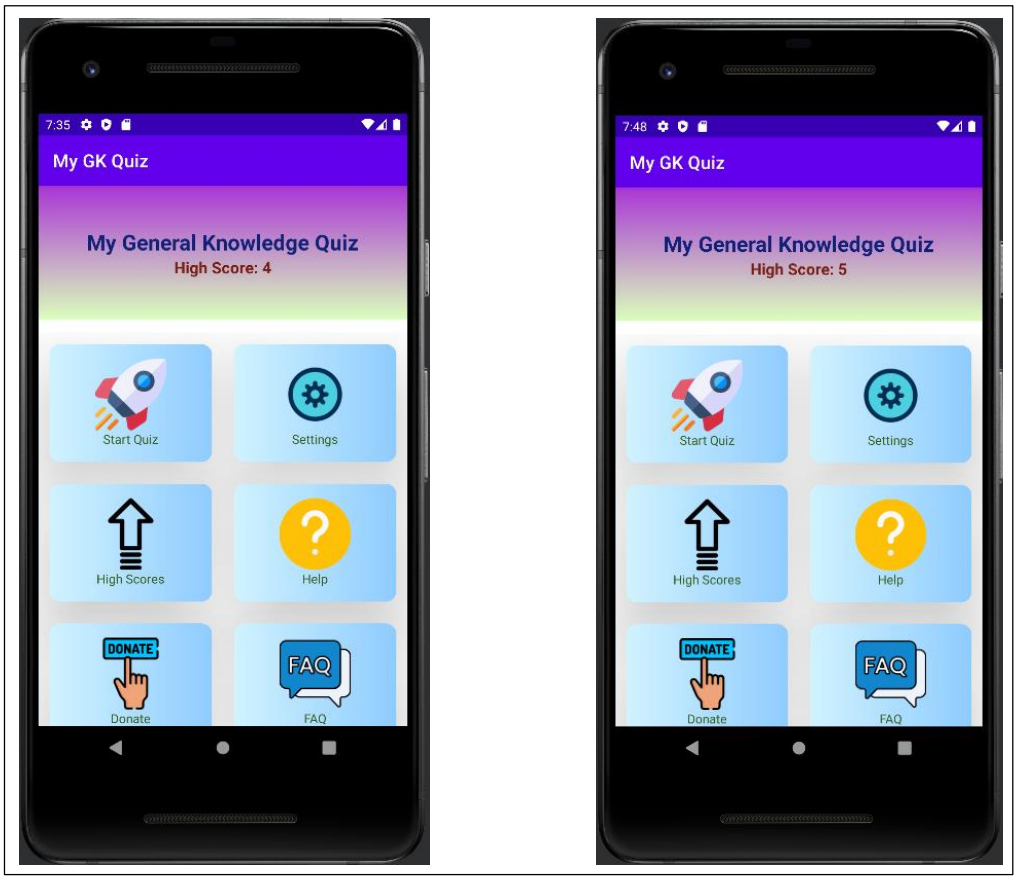


Figure 38 - Testing View high score

Status: Pass.
Post-Conditions: User is able to view highest score.