ANDROID PROGRAMIRANJE

Vežba 7: Izrada interaktivnog kviza i igre memorije

Dodatak: Drag and Drop funkcionalnost

(radi se dve nedelje)

Ove vežba ima cilj da studentima omogući osnovno praktično znanje u razvoju Android aplikacija koristeći Javu i SQLite bazu podataka. Vežba je podeljena na tri dela:

1. **Izrada kviza**: Kreiranje aplikacije za kviz sa SQLite bazom podataka. Pri samostalnoj izradi, temu kviza birate sami. Pitanja mogu iz sporta, kulture, umetnosti, neke oblasti iz gradiva za ovaj ili drugi predmet, bitno je samo da su sadržaji pristojni i relevantni.
2. **Igra memorije**: Razvoj igre memorije korišćenjem osnovnih Android funkcionalnosti. Pri samostalnoj izradi, sličice za igru takođe birate sami. Ideja je da bude grid 4x4, gde vam je potrebno 8 sličica koje mogu ilustrovati životinje, sportove, gradove, logotipe brandova, itd. Vodite računa pri odabiru sa interneta da budu kvadratnog oblika (300x300 px, na primer), da nisu zaštićene autorskim pravima i naravno da su primerene za ovu vežbu.
3. **Drag and Drop funkcionalnost**: Implementacija funkcionalnosti koju ste verovatno viđali u tzv. *Dress-up* igricama, sadrži komponente za povlačenje i ispuštanje elemenata na radnoj površini (ekranu uređaja). Za ovaj deo ne radite samostalno implementaciju, više je informativnog karaktera, da znate kako ovo funkcioniše i kako ga možete potencijalno primeniti u svojim projektima. Vežba će biti ocenjena na osnovu kviza i igre memorije.

Izrada kviza

Android kviz je jednostavna interaktivna aplikacija koja:

- Postavlja korisniku niz od 10 ili više pitanja (višestruki izbor za svako pitanje, četiri opcije).
- Proverava tačnost korisnikovog odgovora.
- Beleži korisnikov skor i prikazuje ga na kraju.

Kviz se koristi za učenje, evaluaciju znanja ili zabavu. Implementacija uključuje elemente poput tekstualnih pitanja, dugmeta za odgovore i mehanizam za praćenje rezultata.

Prvo treba kreirati bazu podataka sa tabelama za korisnike (na početku aplikacije biće log in forma), zatim za pitanja, kao i za igre (svaki korisnik može odigrati jednu ili više igara).

Moj predlog je da aplikacija omogućuje i log in i sign up, gde ćemo imati podatke o korisničkom imenu i šifri, imenu, prezimenu i email adresi, na primer. Drugi podaci za sada nisu potrebni. Za pitanja je potreban tekst, ponuđene opcije, kao i broj opcije koja nosi tačan odgovor. Možete dodati još jednu vrednost koja bi nosila broj poena za to pitanje, ukoliko imate „lakša“ i „teža“ pitanja u svom kvizu, ali to nije obavezno. Igra bi pamtila korisničko ime igrača, datum kada je igrano i koliko je poena osvojio. Inicijalna ideja je da bude *single player* game, ali možete implementirati i mehanizam za dva igrača naizmenično da daju odgovore na pitanja. Isto važi i za igru memorije. Kasnije se to može gledati i kao baza za projekat ukoliko ovaj predmet polažete preko projekta. Napredna verzija ovog koncepta može poslužiti i kao ispitni projekat, svakako.

Što se elemenata samog kviza tiče, tu možemo izdvojiti:

UI elemente

- **TextView**: Prikazuje trenutno pitanje.
- **Button**: Koristi se za opcije odgovora. Može se odraditi i preko radio dugmeta.
- **ProgressBar** (opcionalno): Prikazuje napredak korisnika kroz kviz (npr. pitanje 5 od 10)

Sistem za upravljanje pitanjima

- Pitanja su obično čuvana u nizu ili listi (Array ili ArrayList).
- Svako pitanje ima:
    - Tekst pitanja.
    - Opcije odgovora.
    - Tačan odgovor.

Logiku kviza

- Implementacija uključuje:
    1. Generisanje pitanja redosledom ili nasumično. Predlažem da se pri svakom pokretanju kviza (kada se korisnik uspešno uloguje) generiše deset ili više nasumičnih pitanja iz ukupne kolekcije od 30-40-50 pitanja koja treba staviti u bazu pri kreiranju. Za generisanje ovih tekstova možete koristiti i ChatGPT, samo proverite da li vam daje relevantne informacije, da ne bude besmislenih pitanja.
    2. Proveru da li je korisnik dao tačan odgovor. Aplikacija proverava njegovu izabranu opciju sa tačnom opcijom za to pitanje koju izvlačite iz baze. Ako je odgovor tačan, dodajete poene. Ako je odgovor netačan, možete samo obavestiti korisnika o tome, ili mu oduzeti određen broj poena (npr. minus 3 poena). Još jedna ideja je bojiti polje crvenom ili zelenom bojom nakon odabira odgovora, samo vodite računa da ga resetujete na standardnu boju kada odete na sledeće pitanje.
    3. Ažuriranje trenutnog skora korisnika i prelazak na sledeće pitanje. Skor nije obavezno da se prikaže na ekranu, može tek na kraju kviza izaći kao pop-up prozor. Potreban vam je globalan brojač koji ćete povećavati kada idete na sledeće pitanje. Onog trenutka kada mu je sledeća vrednost veća od broja pitanja, na primer na desetom ste pitanju, više ga ne povećavate, već idete na sekciju za evaluaciju i prikaz rezultata kviza. Posle toga korisnik može početi novu igru ili izaći iz aplikacije. Log out je obavezan. U nastavku je dat deo koda za inspiraciju.

Klasa za upravljanje bazom podataka:

```
public class QuizDbHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "QuizApp.db";
    private static final int DATABASE_VERSION = 1;

    public QuizDbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Metoda za učitavanje svih pitanja iz baze
    public List<Question> getAllQuestions() {
        List<Question> questionList = new ArrayList<>();
        SQLiteDatabase db = this.getReadableDatabase();
        Cursor cursor = db.rawQuery("SELECT * FROM questions", null);
        if (cursor.moveToFirst()) {
            do {
                int id = cursor.getInt(cursor.getColumnIndex("_id"));
                String qText = cursor.getString(cursor.getColumnIndex("question"));
                String opt1 = cursor.getString(cursor.getColumnIndex("option1"));
                String opt2 = cursor.getString(cursor.getColumnIndex("option2"));
                String opt3 = cursor.getString(cursor.getColumnIndex("option3"));
                String opt4 = cursor.getString(cursor.getColumnIndex("option4"));
                int answer = cursor.getInt(cursor.getColumnIndex("answer"));
                Question question = new Question(qText, opt1, opt2, opt3, opt4, answer);
                questionList.add(question);
            } while (cursor.moveToNext());
        }
        cursor.close();
        db.close();
        return questionList;
    }

    // Metoda za snimanje rezultata u tabelu igra
    public void saveGameResult(String username, int points) {
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put("username", username);
        values.put("poeni", points);
        values.put("datum", getCurrentDate());
```

```
        db.insert("igra", null, values);
        db.close();
    }

    // Metoda za dobijanje trenutnog datuma
    private String getCurrentDate() {
        SimpleDateFormat sdf =
                        new SimpleDateFormat("yyyy-MM-dd HH:mm:ss", Locale.getDefault());
        return sdf.format(new Date());
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS questions");
        onCreate(db);
    }
}
```

Model klasa za pitanja:

```
public class Question {
    private String question;
    private String option1;
    private String option2;
    private String option3;
    private String option4;
    private int answer;

   public Question(String qst, String opt1, String opt2, String opt3, String opt4 int answer)
   {
        this.question = qst;
        this.option1 = opt1;
        this.option2 = opt2;
        this.option3 = opt3;
       this.option4 = opt4;
        this.answer = answer;
    }
    // Metoda koja proverava da li je korisnikov odgovor tačan
    public boolean isAnsweredCorrectly(String userAnswer) {
        return userAnswer == answer;
    }
```

```
    public String getQuestion() {
        return question;
    }
    public String getOption1() {
        return option1;
    }

    public String getOption2() {
        return option2;
    }

    public String getOption3() {
        return option3;
    }


    public String getOption4() {
        return option4;
    }

    public int getAnswer() {
        return answer;
    }
}
```

Dinamičko ubacivanje podataka u bazu, ukoliko želite da dodate još neka pitanja naknadno:

```
public void addQuestions() {
    SQLiteDatabase db = getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put("question", "Koji je glavni grad Srbije?");
    values.put("option1", "Novi Sad");
    values.put("option2", "Beograd");
    values.put("option3", "Niš");
    values.put("option4", "Kragujevac");
    values.put("answer", 2);
    db.insert("questions", null, values);
}
```

Interfejs aplikacije:

```
<LinearLayout
```

```
xmlns:android="http://schemas.android.com/apk/res/android"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:orientation="vertical"
android:padding="16dp">

<TextView
    android:id="@+id/textViewQuestion"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textSize="18sp"/>

<RadioGroup
    android:id="@+id/radioGroup"
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <RadioButton
        android:id="@+id/radioButton1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <RadioButton
        android:id="@+id/radioButton2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

   <RadioButton
        android:id="@+id/radioButton4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>

</RadioGroup>

<Button
    android:id="@+id/buttonConfirm"
    android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Potvrdi"/>
</LinearLayout>
```

Logika za prikazivanje pitanja i evaluaciju rezultata:

```
public class MainActivity extends AppCompatActivity {
    private TextView textViewQuestion;
    private RadioButton rb1, rb2, rb3, rb4;
    private RadioGroup radioGroup;
    private Button buttonConfirm;
    private List<Question> questionList;
    private int questionCounter = 0;
    private Question currentQuestion;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        textViewQuestion = findViewById(R.id.textViewQuestion);
        rb1 = findViewById(R.id.radioButton1);
        rb2 = findViewById(R.id.radioButton2);
        rb3 = findViewById(R.id.radioButton3);
        rb4 = findViewById(R.id.radioButton4);
        radioGroup = findViewById(R.id.radioGroup);
        buttonConfirm = findViewById(R.id.buttonConfirm);

        QuizDbHelper dbHelper = new QuizDbHelper(this);
        questionList = dbHelper.getAllQuestions();
        showNextQuestion();
        buttonConfirm.setOnClickListener(v -> {
            if (rb1.isChecked() || rb2.isChecked() || rb3.isChecked() || rb4.isChecked()) {
                checkAnswer();
                showNextQuestion();
            } else {
                Toast.makeText(MainActivity.this, "Izaberite odgovor!",
                               Toast.LENGTH_SHORT).show();
            }
        });
    }
```

```
    private void showNextQuestion() {
        if (questionCounter < questionList.size()) {
            currentQuestion = questionList.get(questionCounter);
            textViewQuestion.setText(currentQuestion.getQuestion());
            rb1.setText(currentQuestion.getOption1());
            rb2.setText(currentQuestion.getOption2());
            rb3.setText(currentQuestion.getOption3());
             rb4.setText(currentQuestion.getOption4());
            questionCounter++;
        } else
            finishQuiz();
    }
    private void checkAnswer() {
        int selectedAnswer = radioGroup.indexOfChild(findViewById
                             (radioGroup.getCheckedRadioButtonId())) + 1;
        if (selectedAnswer == currentQuestion.getAnswer()) {
            Toast.makeText(this, "Tačan odgovor!", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Pogrešan odgovor!", Toast.LENGTH_SHORT).show();
        }
        radioGroup.clearCheck();
    }
    private void finishQuiz() {
        Toast.makeText(this, "Kviz je završen!", Toast.LENGTH_SHORT).show();
        finish();
    }
}
```

U metodi *finishQuiz* kao što je već napomenuto možete dodate i pop-up poruku koja će ispisati broj osvojenih poena i snimiti taj rezultat u bazu podataka. Pamtimo username ulogovanog korisnika, broj poena, kao i datum kada je odigran kviz. Odavde ćemo pozvati pomoćnu metodu koju smo definisali u helper klasi, npr. `dbHelper.saveGameResult(username, totalPoints);`

```
// Metoda za izračunavanje ukupnih poena na osnovu tačnih odgovora
private int calculateTotalPoints() {
    int points = 0;
    for (Question question : questionList) {
        if (question.isAnsweredCorrectly()) points++;
    } return points;
}
```

Igra memorije

Osnovna ideja igre memorije je da kreirate aplikaciju sa matricom 4x4 kartica koje se okreću, a zadatak igrača je da pronađe parove kartica koje se poklapaju. Nakon što odaberete sličice koje će biti u igri, ubacite ih u res/drawable folder (img1.png, img2.png, itd). Dodajte i placeholder sliku koja će se prikazivati dok korisnik ne otvori polje tokom igre. Na početku igre mogu da se mešaju (randomizuju) indeksi u kolekciji sličica i mapiraju na polja koja se otvaraju u igri. Ovo se može postići i na drugačiji način, na primer ubacite sličice kako idu redom a onda ih promešate pomoću Collections.shuffle() metode u Javi. Cilj je svaki put dobiti različit raspored kako bi igrači pravilno koristili tehnike pamćenja i zabavili se u igri. Još jedna ideja je brojati pogotke i skalirati poene prema tome. Na primer, ako je nekom ovo deseti pokušaj da upari sličicu, može dobiti manje poena od onog ko pogađa tek drugi ili treći put. Razmislite o ovome. Moj predlog je skalirati poene na 10, 7 i 5. U nastavku su neki primeri koda.

U activity_main.xml definišite GridLayout za mrežu kartica:

```
<GridLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="4"
    android:rowCount="4"
    android:padding="16dp">
    <!-- Kartice će se generisati dinamički u Java kodu -->
</GridLayout>
```

Klasa MemoryCard koja će predstavljati pojedinačnu karticu:

```
public class MemoryCard {
    private int imageResId;
    private boolean isFlipped;
    private boolean isMatched;

    public MemoryCard(int imageResId) {
        this.imageResId = imageResId;
        this.isFlipped = false;
        this.isMatched = false;
    }
    public int getImageResId() {
        return imageResId;
    }
    public boolean isFlipped() {
        return isFlipped;
    }
```

```
    public void setFlipped(boolean flipped) {
        isFlipped = flipped;
    }

    public boolean isMatched() {
        return isMatched;
    }

    public void setMatched(boolean matched) {
        isMatched = matched;
    }
}
```

Glavna aktivnost:

```
public class MainActivity extends AppCompatActivity {
    private GridLayout gridLayout;
    private List<MemoryCard> memoryCards;
    private MemoryCard firstSelectedCard = null;
    private Button selectedButton1 = null, selectedButton2 = null;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        gridLayout = findViewById(R.id.gridLayout);
        memoryCards = generateCards();
        populateGrid();
    }

    private List<MemoryCard> generateCards() {
        List<MemoryCard> cards = new ArrayList<>();
        int[] images = {
                R.drawable.image1, R.drawable.image2, R.drawable.image3, R.drawable.image4,
                R.drawable.image5, R.drawable.image6, R.drawable.image7, R.drawable.image8
        };

        for (int image : images) {
            cards.add(new MemoryCard(image));
            cards.add(new MemoryCard(image)); // Duplikat za par
        }
```

```
    Collections.shuffle(cards); // Mešanje kartica
    return cards;
}

private void populateGrid() {
    for (int i = 0; i < memoryCards.size(); i++) {
        Button button = new Button(this);
        button.setBackgroundResource(R.drawable.placeholder);
        button.setTag(i); // Čuvamo indeks kartice

        GridLayout.LayoutParams params = new GridLayout.LayoutParams();
        params.width = 0;
        params.height = 0;
        params.rowSpec = GridLayout.spec(i / 4, 1, 1f);
        params.columnSpec = GridLayout.spec(i % 4, 1, 1f);
        button.setLayoutParams(params);

        button.setOnClickListener(v -> onCardClick((Button) v));
        gridLayout.addView(button);
    }
}

private void onCardClick(Button button) {
    int index = (int) button.getTag();
    MemoryCard selectedCard = memoryCards.get(index);

    if (selectedCard.isMatched() || selectedCard.isFlipped()) {
        return;
    }

    button.setBackgroundResource(selectedCard.getImageResId());
    selectedCard.setFlipped(true);

    if (firstSelectedCard == null) {
        // Prva kartica
        firstSelectedCard = selectedCard;
        selectedButton1 = button;
    } else {
        // Druga kartica
        selectedButton2 = button;
        if (firstSelectedCard.getImageResId() == selectedCard.getImageResId()) {
```

```
                // Par je pronađen
                firstSelectedCard.setMatched(true);
                selectedCard.setMatched(true);
                firstSelectedCard = null;
            } else {
                // Nije par, vraćamo kartice nakon kratkog vremena
                new Handler().postDelayed(() -> {
                    selectedButton1.setBackgroundResource(R.drawable.placeholder);
                    selectedButton2.setBackgroundResource(R.drawable.placeholder);
                    firstSelectedCard.setFlipped(false);
                    selectedCard.setFlipped(false);
                    firstSelectedCard = null;
                }, 1000);
            }
        }
    }
}
```

Kada su sve kartice uparene, možete dodati poruku za završetak igre i prikazati broj osvojenih poena. Ukoliko želite da prikazujete i *scoreboard*, tj. tabelu sa korisnicima i poenima, možete nakon svake igre podatke o njoj pamtiti u bazu, tekst, csv ili json fajl. To posle možemo učitavati još na početku i na kraju svake igre ažurirati ga odgovarajućim podacima.

```
private void checkGameCompletion() {
    for (MemoryCard card : memoryCards) {
        if (!card.isMatched()) {
            return;
        }
    }
    Toast.makeText(this, "Čestitamo! Završili ste igru.", Toast.LENGTH_LONG).show();
}
```

Drag and Drop

U ovom odeljku ćemo demonstrirati funkcionalnost *Drag and Drop* u aplikaciji, gde korisnik može da povuče stavke (npr. slike ili tekst) i postavi ih na određena mesta. Iako ovaj deo vežbe nije za ocenu, možete provežbati naredni primer, da dobijete jasniju sliku o celokupnoj ideji i postupku.

Za početak dodajte elemente koji će podržavati funkcionalnost povlačenja i postavljanja. Evo primera za activity_drag_and_drop.xml fajl:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

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

    <!-- Stavka za povlačenje -->
    <ImageView
        android:id="@+id/draggableItem"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:layout_gravity="center_horizontal"
        android:src="@drawable/draggable_image"
        android:contentDescription="Drag Item" />

    <!-- Ciljno područje -->
    <LinearLayout
        android:id="@+id/dropTarget"
        android:layout_width="match_parent"
        android:layout_height="200dp"
        android:layout_marginTop="32dp"
        android:background="@color/teal_200"
        android:gravity="center">

        <TextView
            android:id="@+id/dropText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Drop Here"
            android:textSize="18sp"
            android:textColor="@android:color/white" />

    </LinearLayout>
</LinearLayout>
```

Nakon toga, povežite ImageView za povlačenje i LinearLayout kao ciljnu oblast za postavljanje.

```
public class DragAndDropActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_drag_and_drop);
        ImageView draggableItem = findViewById(R.id.draggableItem);
```

```
    LinearLayout dropTarget = findViewById(R.id.dropTarget);
    // Postavljanje funkcionalnosti
    setupDragAndDrop(draggableItem, dropTarget);
}

private void setupDragAndDrop(ImageView draggableItem, LinearLayout dropTarget) {
    draggableItem.setOnLongClickListener(v -> {
        ClipData clipData = ClipData.newPlainText("DragData", "Item");
        View.DragShadowBuilder shadowBuilder = new View.DragShadowBuilder(v);
        v.startDragAndDrop(clipData, shadowBuilder, v, 0);
        return true;
    });

    dropTarget.setOnDragListener((v, event) -> {
        switch (event.getAction()) {
            case DragEvent.ACTION_DRAG_STARTED:
                // Povlačenje započeto
                return true;

            case DragEvent.ACTION_DRAG_ENTERED:
                // Stavka je ušla u ciljnu oblast
              dropTarget.setBackgroundColor(getResources().getColor(R.color.teal_700));
                return true;

            case DragEvent.ACTION_DRAG_EXITED:
                // Stavka je izašla iz ciljne oblasti
              dropTarget.setBackgroundColor(getResources().getColor(R.color.teal_200));
                return true;

            case DragEvent.ACTION_DROP:
                // Povlačenje i postavljanje završeno
                View draggedView = (View) event.getLocalState();
                dropTarget.removeAllViews();
                dropTarget.addView(draggedView);
                Toast.makeText(this, "Item Dropped!", Toast.LENGTH_SHORT).show();
                return true;

            case DragEvent.ACTION_DRAG_ENDED:
                // Povlačenje je završeno
```

```
                    dropTarget.setBackgroundColor(getResources().getColor(R.color.teal_200));
                    return true;
                default:
                    return false;
            }
        });
    }
}
```

Testiranje

1. Pokrenite aktivnost i proverite da li ImageView može da se povuče i postavi u ciljnu oblast.
2. Tokom povlačenja proverite promenu boje ciljne oblasti (ACTION_DRAG_ENTERED i ACTION_DRAG_EXITED).
3. Ako je stavka pravilno postavljena, prikazati poruku kao potvrdu.

Korisni resursi

Kviz:

https://www.youtube.com/watch?v=ELuEIPRksPo&ab_channel=BamideleOguntuga

https://www.youtube.com/playlist?list=PLvfk4EH3FLEezdUSC0eLHI14tVZ7xGq9u

https://www.youtube.com/watch?v=9i8-64ZxSnk&ab_channel=RunCodes

https://www.youtube.com/playlist?list=PLBcjj47fhbjXhYscBCEBY8-NLhUey4Rsm

Najbolje je postavljene klipove i playliste gledati bez tona, ili sa ugrađenim titlovima, s obzirom na razumevanje engleskog jezika kojim govore predavači. Uprkos tome, tutorijali su tehnički dobri.

Igre memorije:

https://www.youtube.com/watch?v=B7--mnjOOI4&ab_channel=SkyFish

https://www.youtube.com/watch?v=U4Wtjewy7EY&ab_channel=RahulPandey

https://www.youtube.com/watch?v=94CWNE9ruMA&ab_channel=TihomirRadev

Svi su tutorijali zanimljivi i korisni na svoj način, ali obratite pažnju da klip Tihomira Radeva i slobodno optimizujte to rešenje u svojoj izvedbi. Svakako ne mora toliko stvari da se kopira i kuca.

Drag and Drop:

https://www.youtube.com/watch?v=8hzJNjj8Zck&ab_channel=PhilippLackner

Implement drag and drop with views | Views | Android Developers

Android - Drag and Drop