Semestrálna práca

Vývoj aplikácií pre mobilné zariadenia

Dokumentácia

Meno a priezvisko:	Matúš Kasák
Študijná skupina:	5ZYI24
Akademický rok:	2022/2023
Semester:	letný

Názov aplikácie - **GeoGuess**

❖ Popis a analýza problému

➢ Špecifikácia zadania

- Aplikáciu je určená pre používateľov, ktorí sa chcú otestovať koľko krajín dokážu spamäti napísať a koľko im to bude trvať.
- Podobné aplikácie sú dostupné iba na webových stránkach ako napríklad JetPunk.

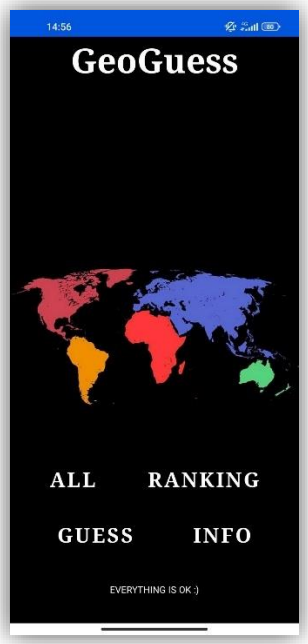
➢ Rozdiel v podobných aplikáciách

- Na Obchod Play nájdeme desiatky aplikácií, ktoré sa snažia formou kvízu naučiť používateľa informácie o svete ako hlavné mestá, vlajky, počet obyvateľov a pod. na ktoré je možné odpovedať.
- Takmer v každom prípade je položená otázka a používateľ si buď vyberá z odpovedí, napíše vlastnú odpoveď alebo vyberá na obrázku.
- Niektoré ale neponúkajú ani možnosť si pozrieť informácie priamo v aplikácii a tak sa ich naučiť.
- Aplikácia neobsahuje iba nezávislé krajiny.

❖ Návrh riešenia problému

➢ Prípady použitia

-



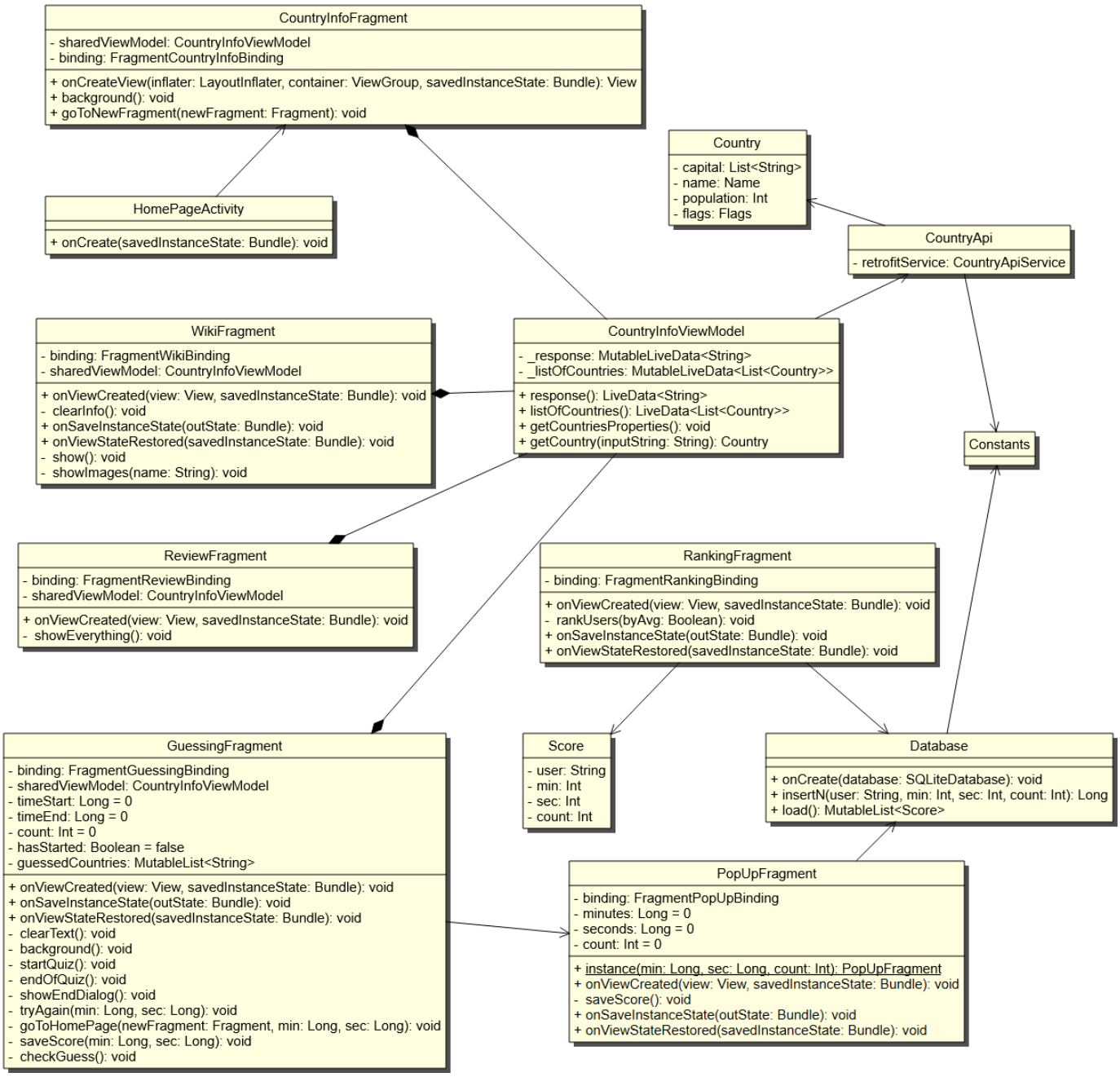
Na *obrázku 1* vidíme domovskú obrazovku. Používateľ si má možnosť vybrať z 4 tlačidiel.

- ALL – zobrazenie všetkých štátov spolu s hlavnými mestami
- RANKING – celkové hodnotenie hier, možnosť vybrať podľa čoho zoradí

- GUESS – *obrázok 2*
    - Samotný kvíz, obsahuje tlačidlá na spustenie, ukončenie kvízu a tlačidlo hádania
    - Po ukončení sa zobrazia informácie o hre a možnosť uloženia dosiahnutého skóre
- INFO – *obrázok 3*
    - Informácie o krajine

➢ Návrh aplikácie

- Niektoré triedy som vynechal kvôli prehľadnosti, hlavne také ktoré iba dopĺňajú triedu Country, takisto parametre Country som dal iba pár



❖ **Popis implementácie**

- Aplikácia obsahuje aktivitu HomepageActivity ktorá obsahuje *FragmentContainerView*. Všetky fragmenty reagujú správne na otočenie zariadenia uložením si stavu a následnom načítaní
- Ako prvý sa zobrazí fragment CountryInfoFragment.
  - Vytvorí sa inštancia *CountryInfoViewModelu* pomocou lazy, inicializuje sa až pri prvom prístupe

```
private val sharedViewModel: CountryInfoViewModel by lazy {
    ViewModelProvider(requireActivity()).get(CountryInfoViewModel::class.java)
}
```

  - Kliknutím na tlačidlá sa zavolá metóda s príslušným fragmentom v parametri na zobrazenie ďalšieho fragmentu. addToBackStack slúži na zapamätanie si na aký fragment sa posunúť stlačením späť

```
private fun goToNewFragment(newFragment: Fragment) {
    val fragmentManager = requireActivity().supportFragmentManager
    val fragmentTransaction = fragmentManager.beginTransaction()
    fragmentTransaction.replace(R.id.mainFragmentContainerView, newFragment)
    fragmentTransaction.addToBackStack( name: null)
    fragmentTransaction.commit()
}
```

- CountryInfoViewModel je trieda, ktorá uchováva dáta o krajinách
  - Funkcia *getCountriesProperties* načíta do atribútu *_listOfCountries* LiveData z CountryApiService
  - Načítanie prebehne *asynchrónne*, čiže užívateľské prostredie nečaká kým sa dokončí.

```
fun getCountriesProperties() {
    viewModelScope.launch { this: CoroutineScope
        try {
            _listOfCountries.value = CountryApi.retrofitService.getProperties()
            _response.value = "EVERYTHING IS OK :)"
        } catch (e: Exception) {
            _response.value = "FAILURE: ${e.message} \nIT IS NOT POSSIBLE TO PLAY!"
        }
    }
}
```

- Funkcia *getCountry*(*inputString: String*) vráti objekt Country, ktorej bežné meno sa zhoduje s parametrom.
- Pomocou DataBindingu vypíšeme informáciu o úspešnom/neúspešnom načítaní dát.

```
<data>
    <variable
        name="viewModel"
        type="com.example.geoguess.activities.CountryInfoViewModel" />
</data>
```

```
android:text="@{viewModel.response}"
```

- Na načítanie som použil JSON, ktorý som z url adresy pomocou *retrofit* a *moshi* prekonvertoval na Kotlin objekty.

```
interface CountryApiService {
    Kasinhou
    @GET(Constants.ADD_URL_ADDRESS)
    suspend fun getProperties():
            List<Country>
}
val retrofitService : CountryApiService by lazy {
    retrofit.create(CountryApiService::class.java)
```

- Vytvoril som data triedu Country, ktorá má parametre zodpovedajúce Json objektom. Niektoré parametre boli komplexnejšie – hashmapy,... tak som podľa potreby vytvoril ďalšie data triedy.
  - Spravil som funkciu, ktorá vracia upravenú textovú reprezentáciu o informáciach o krajine, ktoré sa vypisujú v WikiFragment
- WikiFragment tiež vytvorí inštanciu *ViewModelu*
  - Funkcia show zistí čo zadal používateľ a ak nájde takú krajinu, zobrazí o nej informácie, vlajku a erb
  - Obrázky som načítaval pomocou knižnice *Glide*, ktorá ich načítala pomocou url adresy, ktorá je definovaná ako parameter v data triede Country. Keď je url adresa chybná zobrazí sa obrázok ic_broken_image. Počas načítavania je zobrazená animácia točiaceho sa koliečka

```
Glide.with( fragment: this) RequestManager
    .load(flagUrl) RequestBuilder<Drawable!>
    .placeholder(R.drawable.loading_animation)
    .error(R.drawable.ic_broken_image)
    .into(binding.ivFlag)
```

- ReviewFragment obsahuje funkciu *showEverything*, ktorá najskôr načíta do zoznamu názvy a hlavné mestá všetkých štátov, utriedi ho podľa abecedy a následne vypíše.
- GuessingFragment uchováva informácie o kvíze (počet uhádnutých, čas, zoznam uhádnutých...)
  - Obsahuje funkciu *checkGuess*, ktorá sa zavolá ak používateľ klikne na tlačidlo TRY
  - Zistí sa názov krajiny a porovná sa v zozname či taká existuje. Ak existuje a ešte nie je v zozname uhádnutých tak sa tam pridá a aktualizuje sa počet, zoznam a zasvieti sa nápis CORRECT. Ak krajina existuje ale už sa v zozname nachádza, atribúty sa neaktualizujú. Ak krajina neexistuje, zasvieti sa WRONG
  - Po kliknutí na tlačidlo END QUIZ alebo po uhádnutí všetkých krajín sa zavolá funkcia *endQuiz*
    - Všetko sa vyčistí, stopne sa čas a zobrazí sa dialógové okno s informáciami o hre a možnosťou ukončiť hru alebo skúsiť znova
    - V oboch prípadoch má používateľ možnosť uložiť si svoj výsledok

```
private fun saveScore(min: Long, sec: Long) {
    val popUpWindow = PopUpFragment.instance(min, sec, count)
    popUpWindow.show((activity as AppCompatActivity).supportFragmentManager, tag: "Pop Up Window")
}
```

    - PopUpFragment má funkciu *instance*, ktorá vráti inštanciu fragmentu a priradia sa hodnoty parametrov k atribútom

```
companion object {
    Kasinhou
    fun instance(min: Long, sec: Long, count: Int) : PopUpFragment {
        val newFragment = PopUpFragment()
        newFragment.minutes = min
        newFragment.seconds = sec
        newFragment.count = count
        return newFragment
    }
}
```

    - Funkcia *saveScore* zistí zadané meno od používateľa a uloží atribúty meno, minúty, sekundy, počet do databázy pomocou funkcie *insertN*

```
(Database(requireContext()).insertN(binding.nameToSave.text.toString(), minutes.toInt(), seconds.toInt(), count).
```

- Trieda Database dedí SQLiteOpenHelper obsahuje funkciu *onCreate*, ktorá sa zavolá práve raz, a to vtedy keď databáza nie je ešte vytvorená
- Všetky názvy použité v Database sú uložené ako konštanty v Constants
  - Ďalej má 2 funkcie, *insertN* a *load*
  - InsertN sa pokúsi vložiť parametre ako atribúty do databázy, ak sa vyskytne nejaký problém ako napríklad porušenie unikátnosti primárneho kľúča funkcia vráti -1 a vytvorí sa nová inštancia PopUpFragmentu
  - *Load* prečíta všetky riadky tabuľky v databáze a uloží ich do zoznamu objektov Score, ktorý aj vráti

- RankingFragment má funkciu *rankUsers* ktorá na základe zadaného parametru typu boolean zoradí riadky databázy buď podľa počtu uhádnutých krajín alebo podľa priemeru na jedno uhádnutie. Celkové poradie sa dá preklikávať pomocou tlačidiel
  - Dáta si ukladá do MutableMap<Double, String> kde kľúč je hodnota podľa ktorej zoraďuje

```
rankings[((it.sec + (it.min * 60.0)) / it.count)] =
    "${it.user}\n     time: ${it.min}min and ${it.sec}sec, guessed: ${it.count}"
```

- XML súbory
  - Použil som ScrollView pri fragment_guessing, fragment_ranking, fragment_review a fragment_wiki kvôli tomu že zobrazenie môže presiahnuť obrazovku a tým pádom by nebolo možné zobraziť všetky informácie, komponenty
  - Rozloženie som používal najčastejšie ConstraintLayout, ale takisto som použil aj LinearLayout
  - UI sa skladá z Button, TextView, EditText, ImageView

Pracoval som sám, používal som práce z cvičení, codelaby a prednášky. Internet som tiež používal, ale iba na vysvetlenie a naštudovanie si informácii. Informácie hlavne o vytvorení databázy a načítavaní json objektov.