Technická univerzita v Liberci

Ekonomická fakulta

Semestrální projekt

Programování mobilních aplikací

Jméno studenta: Pavel Švec

Ročník: 1.

Akademický rok: 2023/2024

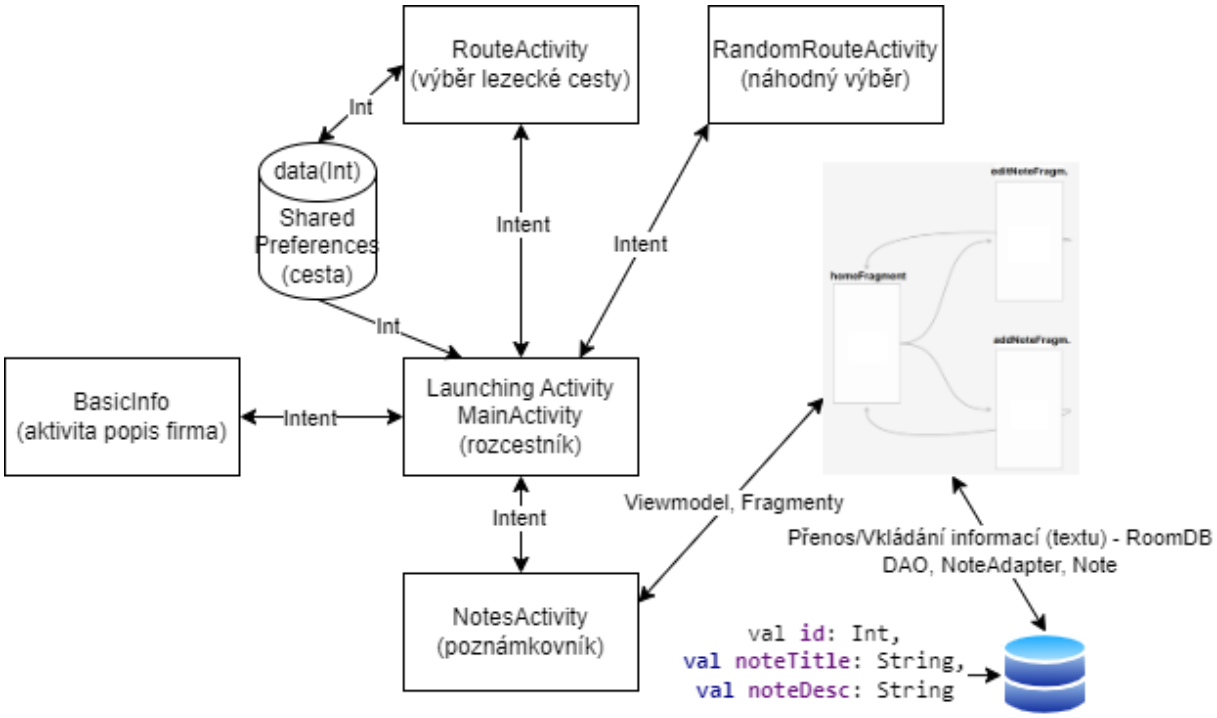
Datum vypracování: 18.1.2024

Obsah

Seznam ilustrací (obrázků)

1 Stavba aplikace

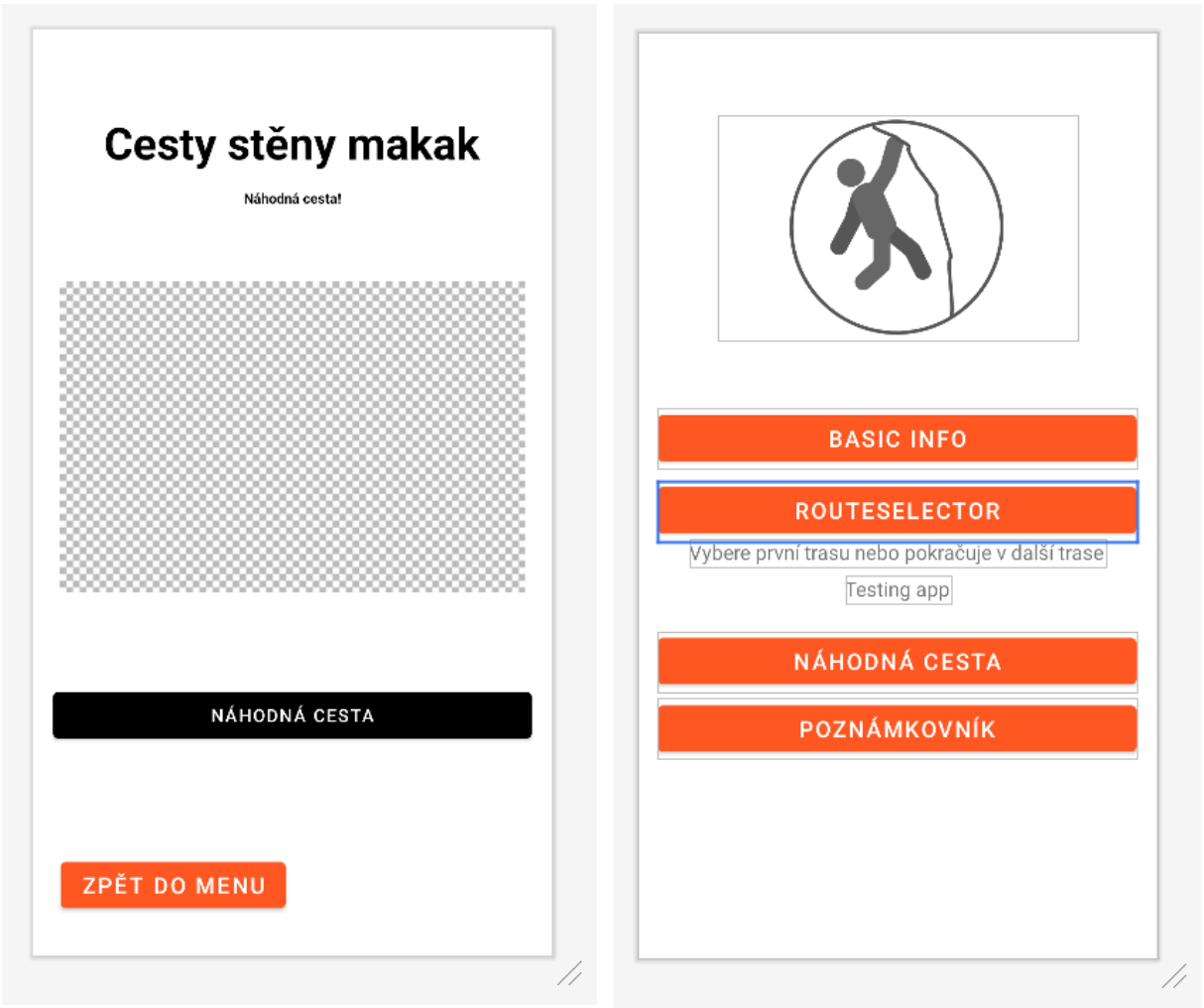
V rámci tohoto předmětu byla vyvinuta aplikace využívající metody a postupy probírané během výuky, ale také používá jiné metody a postupy. Samotná aplikace má několik základních funkcí. Zprvopočátku se mělo jednat a aplikaci pro lezce v Jablonecké vnitřní hale na lezení, ale nakonec jsem se zaměřil spíše na tvorbu a naučení se několika různých funkcí. Součástí aplikace tak nakonec je rozcestník nabízející přesun mezi aktivitami na výběr lezecké cesty (postupně) s uložením poslední vybrané trasy pro příští otevření aplikace, na výběr náhodné cesty, neinteraktivní aktivita zobrazující základní informace o firmě, a nakonec plně funkční poznámkový blok pro zaznamenání informací (například na příští návštěvu podniku). Na následujícím schématu jsou ukládané datové údaje znázorněné pomocí cylindrů.



Obrázek 1 Jednoduché schéma složení aplikace

Po spuštění aplikace se zobrazí *MainActivity*, která slouží jako rozcestník do dalších aktivit. Také zobrazuje údaje o poslední zdolávané cestě v rámci funkcionality *RouteSelector*. Aktivitu doplňuje grafický element z veřejné knihovny animací *Lottie* ve formátu *.json*. Údaje o jednotlivých cestách jsou uloženy pomocí *array* (datového pole). Díky indexování údajů o posledním zobrazovaném obrázku cesty a funkcionalitě *SharedPreferences* je možné tak v textu oznámit uživateli poslední vybranou trasu.

Objekt *SharedPreferences* ukazuje na soubor obsahující dvojice klíč-hodnota a poskytuje jednoduché metody pro jejich čtení a zápis. V rámci aplikace se využívá zápisu do *SharedPreferences* a také čtení z nich. Díky tomu se při každém otevření aplikace zobrazí v textovém poli pod tlačítkem *RouteSelector* poslední vybraná cesta (Android Studio Developers, 2023).

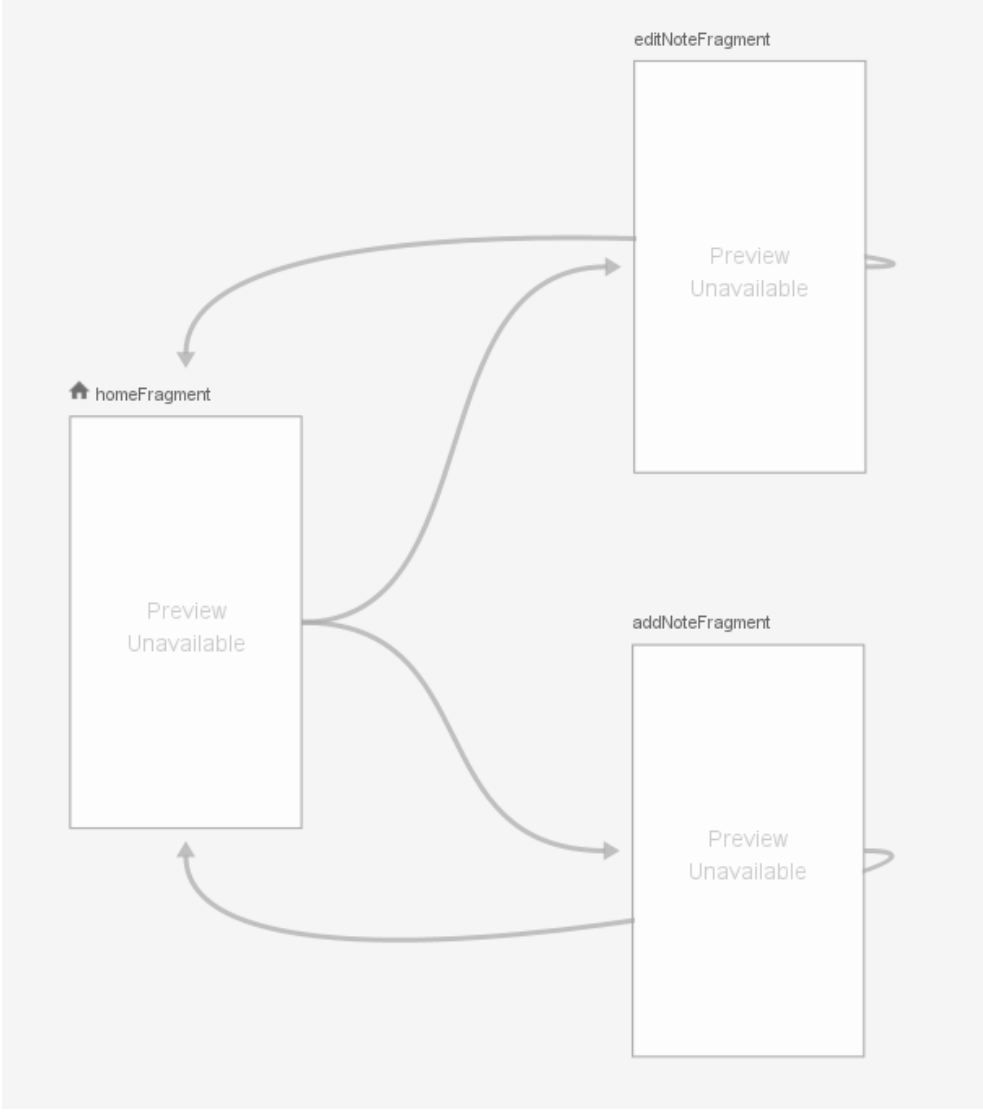


Obrázek 2 Náhled na rozhraní RandomRouteActivity a MainActivity

Tlačítko *RouteSelector* odkazuje na aktivitu *RouteActivity*, kde pak uživatel postupně prochází cesty od nejlehčí po nejtěžší. Uživatele informuje o vybrané cestě informace na obrázku a text, který je díky funkci *mapOf* převádějící informaci o zobrazovaném obrázku v text zobrazen. Tlačítko *Náhodná cesta* odkazuje na *RandomRouteActivity*, kde je díky funkci *random()* zobrazována náhodně vybraná cesta.

2 Využití Room databáze a poznámkový blok

Tlačítko *Poznámkovník* spouští aktivitu *NotesActivity.* V rámci této aktivity se používá jiného přístupu k zobrazování layoutu. Využívá se třídy *ViewModel,* která pomocí funkce setupViewModel() v této aktivitě vytváří layout. Funkce je navázána na *NoteViewModelFactory,* která slouží jako továrna na vytvoření instancí *ViewModelu.* Samotný *ViewModel* je pak navázaný na repozitář *NoteRepository,* který je propojený s *NoteDao.* V rámci *NoteRepository* se nachází *suspend* funkce (mohou být pozastaveny a vyvolány) obsahující db.getNoteDao(), čímž dochází k interakci s DAO a databází. Přesný vzhled tabulky databáze je stanoven v *Note.* Samotný vzhled a funkce tlačítek jsou definované v dílčích fragmentech.



Obrázek 3 Fragment Container v NotesActivity, stanovení provázání v nav_graph.xml

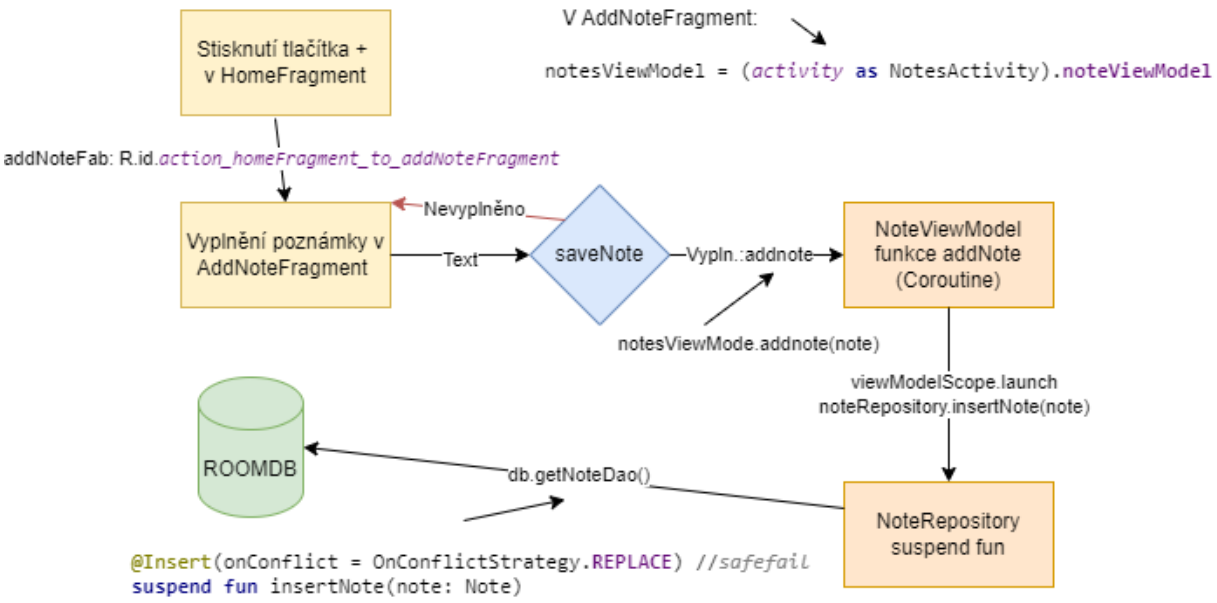
Pro spuštění *suspend* funkcí v *NoteRepository* je využito řešení *Coroutines,* kdy v *NoteViewModel* jsou definovány tyto funkce:

```
class NoteViewModel(app: Application, private val noteRepository: NoteRepository):  AndroidViewModel(app) {

    Pavel Švec
    fun addNote(note: Note) =
        viewModelScope.launch { this: CoroutineScope  //coroutine - prevence potencial memory leaks
            noteRepository.insertNote(note)
        }

    Pavel Švec
    fun deleteNote(note: Note) =
        viewModelScope.launch { this: CoroutineScope  //coroutine - prevence potencial memory leaks
            noteRepository.deleteNote(note)
        }

    Pavel Švec
    fun updateNote(note: Note) =
        viewModelScope.launch { this: CoroutineScope  //coroutine - prevence potencial memory leaks
            noteRepository.updateNote(note)
        }

    Pavel Švec
    fun getAllNotes() = noteRepository.getAllNotes()

    Pavel Švec
    fun searchNote(query: String?) =
        noteRepository.searchNote(query)
}
```

Obrázek 4 Využití Coroutines pro zprovoznění interakce s DB

Díky tomu je tak možné poznámky přidávat, mazat, upravovat, ale také vyhledávat a všechny zobrazit. Bohužel ale vyhledávací funkce není plně spolehlivá, je nutné u ní udělat úpravy, pokud by se na projektu pracovalo dále. V rámci *NoteDAO* (Database Access Object) jsou pak definované úkony *insertNote* (@Insert), *updateNote* (@Update), *deleteNote*(@Delete) a dva výběry (@Query). Vzhledem k několika operacím s databází, a především fragmentace aplikace, je poměrně obtížné rozepsat veškeré interakce s databází v rámci všech fragmentů. Asi nejdůležitější částí je samotné přidávání poznámek, a to je znázorněno na další ilustraci. Tři základní fragmenty mají následující funkce:

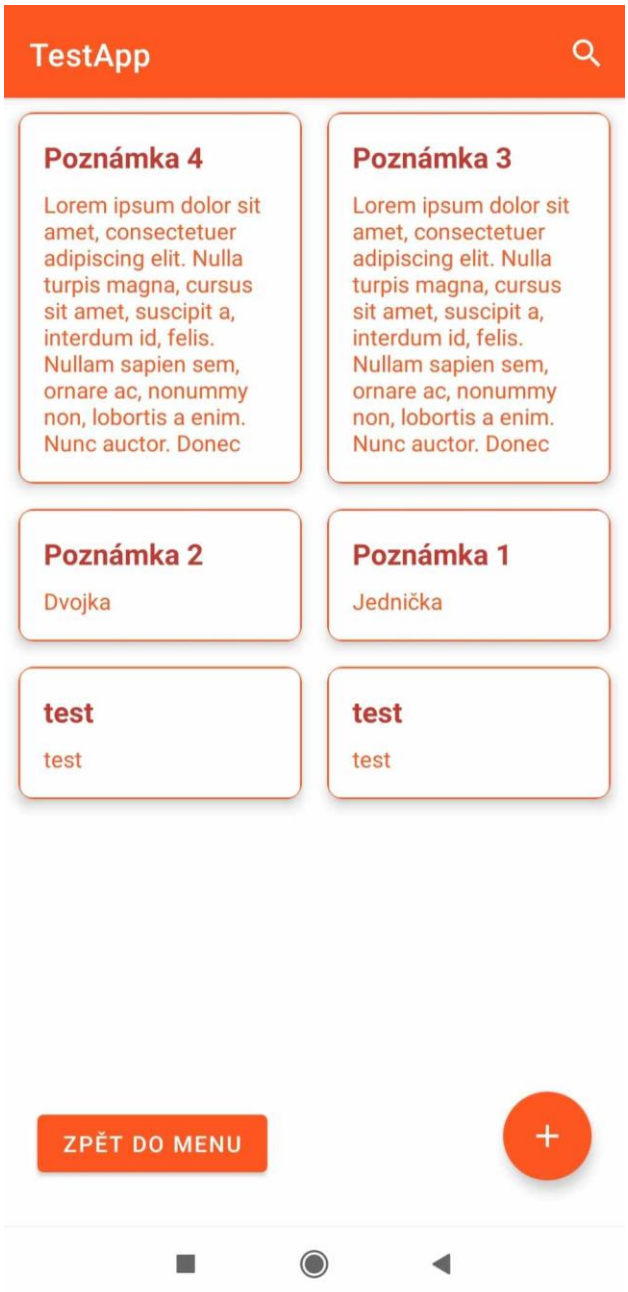
- *AddNoteFragment*: ukládání nové poznámky – kontrola vložení textu (toasty),
- *EditNoteFragment*: úprava poznámky, odstranění poznámky (toasty),
- *HomeFragment*: místo na přesun do *MainActivity*, hlavní fragment, zobrazení obrázku šipky při prázdné DB poznámek, vyhledávací funkce, zobrazení poznámek.



Obrázek 5 Schéma postupu přidání nové poznámky

Samotná databáze *NoteDatabase* je definovaná pomocí tagu @Database, po stanovení funkce getNoteDao(): *NoteDao* je pak také stanoven interface s tagem @Dao, kde jsou definovány příkazy. Databáze je také chráněna kódem pro uzavření databáze pro práci pouze s jednou instancí. Samotná struktura tabulky, která je uložená v DB je stanovena již dle zmíněného *Note*. Zde je nastavena tvorba primárního klíče na *autoGenerate*.

Interakce s UI je zajištěna pomocí adaptéru *NoteAdapter*. Pro správné zobrazení poznámek je nutné zavést několik funkcí. V adaptéru se také nachází posluchač pro navigaci na *EditNoteFragment* dle zvolené položky. Pro zobrazení poznámek ve *ViewHolder* je použita funkce *onBindViewHolder,* která pomocí *binding* metody doplňuje název a popis poznámek v holderu. Adaptér také obsahuje kontrolu na porovnání položek v seznamu pro případ identičnosti. V návaznosti na *NoteAdapter* se pak samotné UI zobrazuje v *HomeFragment,* kdy pomocí *binding* metody je použit *layoutManager* (konkrétně *StaggeredGridLayoutManager* – tvorba 2 sloupců pro rozložení položek). V *HomeFragment* je také zabudovaný *observer,* který slouží pro aktualizaci UI při změně dat.



Obrázek 6 Poznámkový blok v NotesActivity

Závěr

Po ruční kontrole jednotlivých aktivit na připojeném mobilním zařízení se podařilo ověřit základní implementované funkce aplikace. Aplikace zatím nepadá, ale stále zůstává otázkou mnohé vylepšení pro snadnější práci s UI a samotná funkcionalita stanovená kódem. Pro kontrolu funkčnosti *Room* databáze je použito funkce App Inspection v rámci rozhraní Android Studio. Po inspekci je možné vidět již vložené poznámky, databáze funguje. Do budoucna je možné stanovit lepší směr vývoje aplikace a nesoustředit se pouze na funkcionality, ale na samotné využití aplikace.

Zdroje

Android Studio Developers, 2023. *Kotlin coroutines on Android*. Developer.android.com [cit. 2024-1-18]. Dostupné z: https://developer.android.com/kotlin/coroutines

Android Studio Developers, 2023. *Save data in a local database using Room.* Developer.android.com [cit. 2024-1-18]. Dostupné z: https://developer.android.com/training/data-storage/room

Android Studio Developers, 2023. *Save simple data with SharedPreferences*. Developer.android.com [cit. 2024-1-18]. Dostupné z: https://developer.android.com/training/data-storage/shared-preferences

Příloha

Příloha 1 Kontrola databáze připojeného zařízení v App Inspection v rozhraní Android Studio