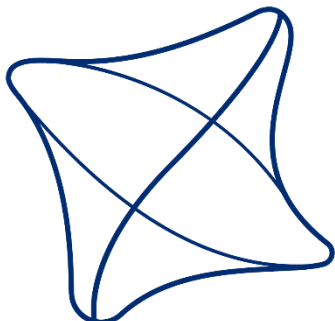


**ŽILINSKÁ UNIVERZITA V ŽILINE**  
**FAKULTA RIADENIA A INFORMATIKY**



Vývoj aplikácií pre mobilné zariadenia

*Názov semestrálnej práce:*

*„Poznámky“ – mobilná aplikácia pre Android zariadenia*

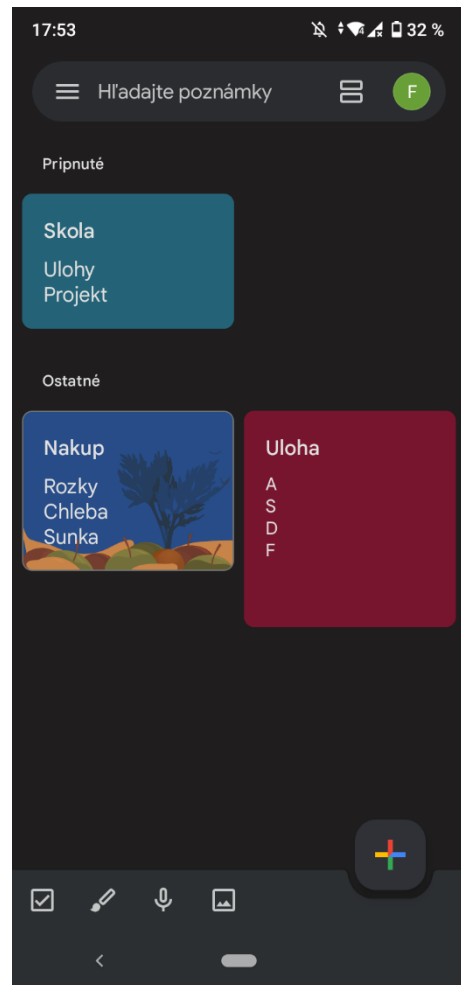
*Vypracoval: Filip Masaryk*  
*Školský rok: 2021/2022*  
*Študijná skupina: 5ZYR23*

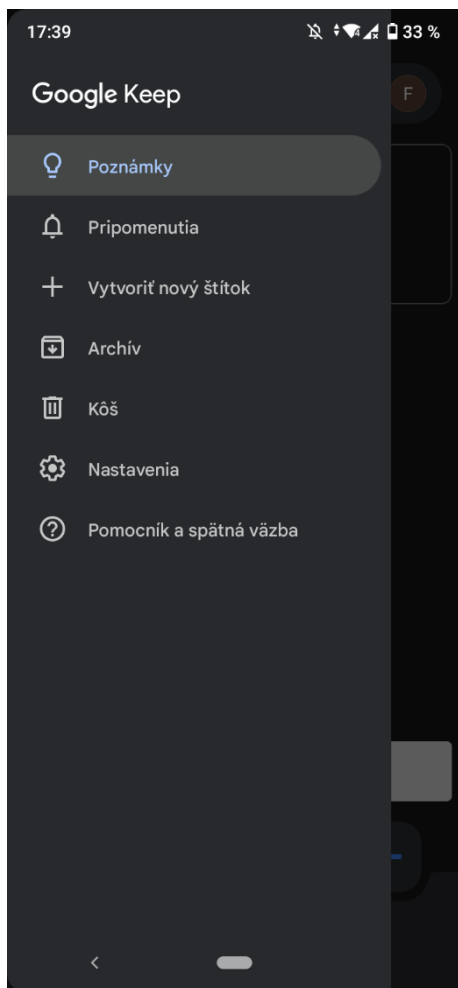
## 1. Popis a analýza riešeného problému

Úlohou bolo navrhnuť aplikáciu pre Android v ktorej si užívateľ môže prehliadať svoje poznámky. Aplikácia umožňuje užívateľovi pridať, vymazať poznámky a taktiež si ich môže „pripnúť“, čo zabezpečí aby sa zobrazovali úplne na samom vrchu. Každá poznámka obsahuje meno a popis. V aplikácii je taktiež možné vyhľadávať medzi poznámkami podľa mena. Poznámky sú ukladané v Android Room databáze.

## 2. Spracovanie prehľadu podobných aplikácií

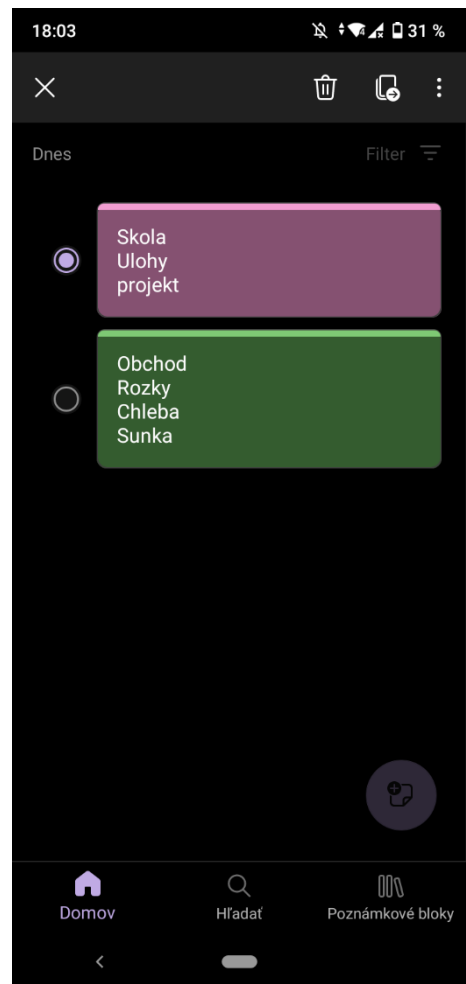
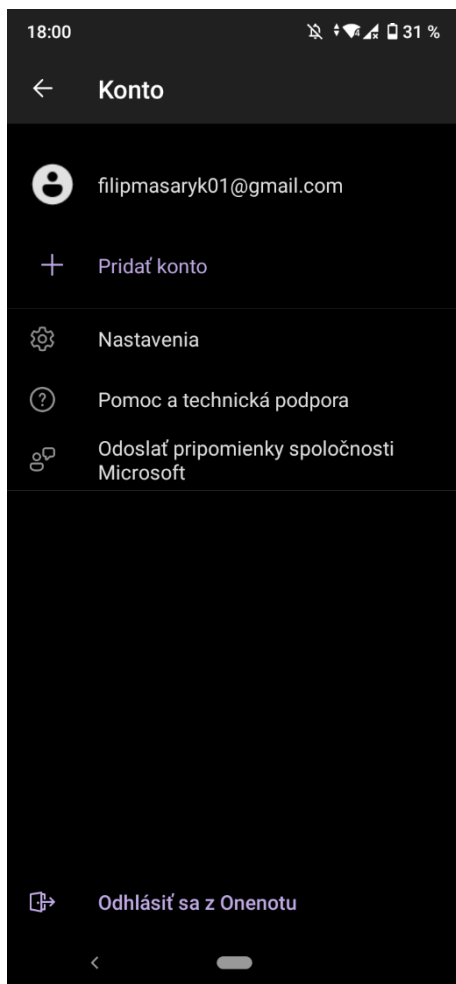
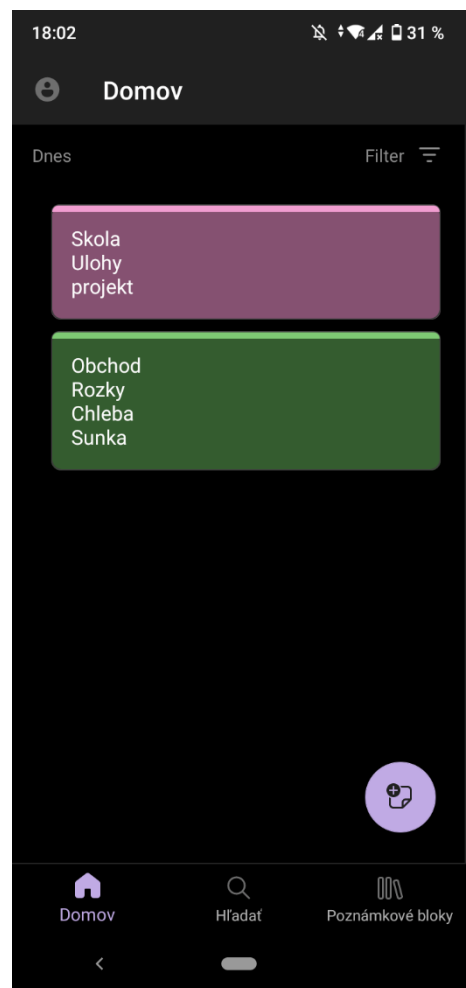
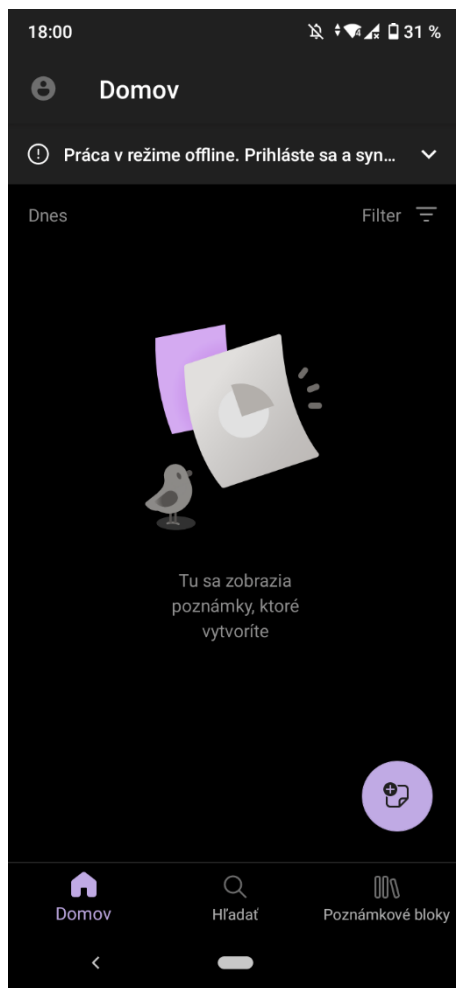
### Google Keep





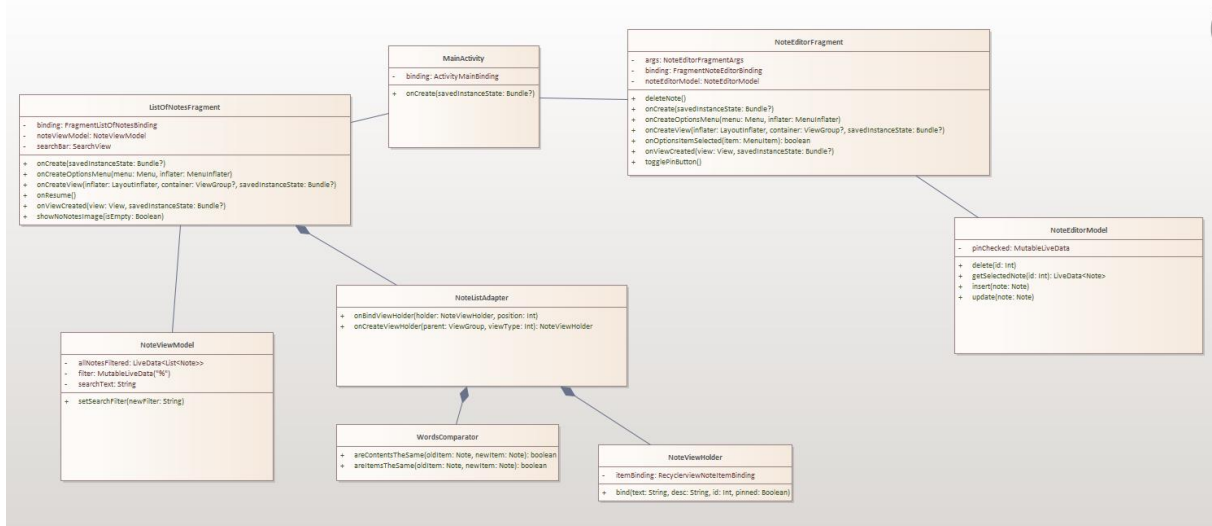
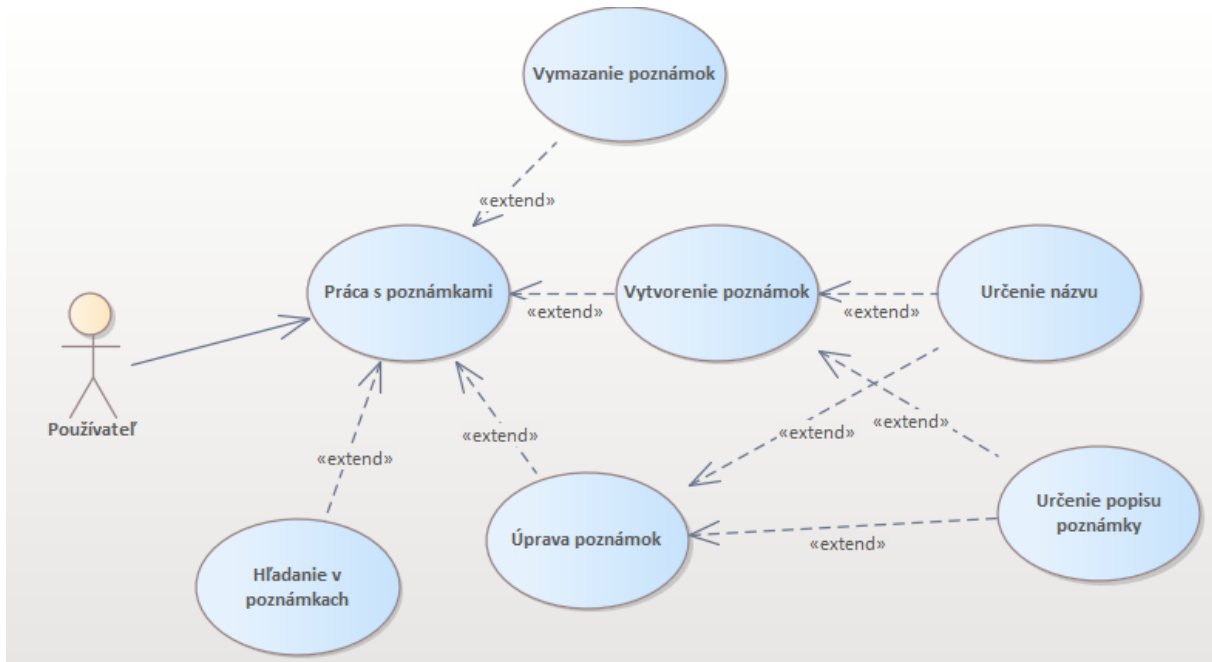
Google Keep je najpopulárnejšia aplikácia na poznámky. Do aplikácie sa dá prihlásiť pomocou google účtu na ktorom je možné si uchovávať dané poznámky. Okrem obvyčajnej funkcie pridania/odstránenia textových poznámok sa tu taktiež dajú vytvoriť poznámky pomocou kreslením na plátno. Google Keep ponúka možnosť vytvorenia si poznámok s rôznymi farebnými pozadiami, a taktiež je možné si zvoliť pozadie ktoré si môžete odfoťiť fotoaparátom alebo nahráť fotku z galérie mobilu.

## Microsoft OneNote



Microsoft OneNote je taktiež jedna z populárnejších aplikácií na poznámky. Táto aplikácia obsahuje menej funkcií ako tá od Googlu. Všimol som si že OneNote neobsahuje kôš, čo znamená že odstránené poznámky sú natrvalo vymazané a už niet možnosti sa k danej poznámke vrátiť. Taktiež v tejto aplikácii sa nedá určiť si pripomienky/upozornenia na dané poznámky. Celkovo je táto aplikácia oveľa jednoduchšia ako Google Keep.

### 3. Návrh riešenia problému



## 4. Popis implementácie

V aplikácii využívam Single Activity architektúru, ktorá znamená, že v aplikácii je prítomná len jedna aktivita v ktorej sa pomocou **NavigationComponentu** striedajú viaceré fragmenty. Vďaka tomuto prístupu sa dokážu ľahšie znovu používať jednotlivé fragmenty a jednoduchšie sa definujú prechody medzi nimi vďaka **NavigationComponentu**.

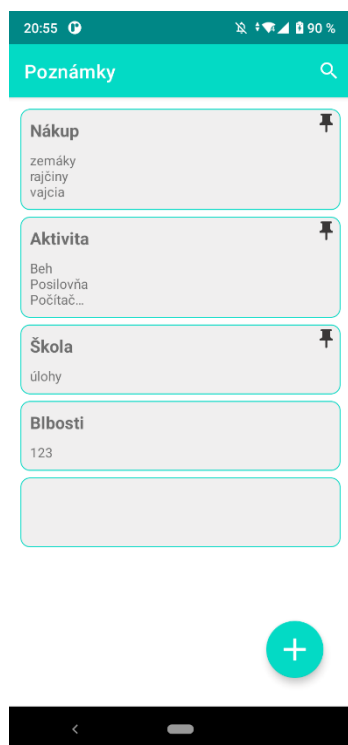
Na dizajnovanie jednotlivých fragmentov využívam knižnicu **com.google.android.material**, pretože obsahuje už pripravené a nadizajnované komponenty, ktoré sa riadia pravidlami Material Designu.

V aplikácii mám dokopy 1 aktivitu a 2 fragmenty:

**ListOfNotesFragment** – Fragment, ktorý zobrazuje všetky poznámky, po kliknutí na nejakú poznámku sa presmeruje na NoteEditorFragment do ktorého sa vyplnia dáta kliknutej poznámky. Po stlačení tlačítka vpravo dole sa tiež presmeruje na NoteEditorFragment kde môžeme pridať novú poznámku.

Toto zobrazenie je implementované pomocou RecyclerView a k nemu prislúchajúcemu adaptéru. Adaptér ma v sebe taktiež implementovanú triedu DiffUtils, ktorá porovnáva zoznam poznámok, ktorý sa už v adaptéri nachádza a novo priradeným zoznamom a vyhodnocuje prvky zoznamu, ktoré sa nemenia a tým pádom zabraňuje zbytočnému prekresľovaniu prvkov na obrazovke.

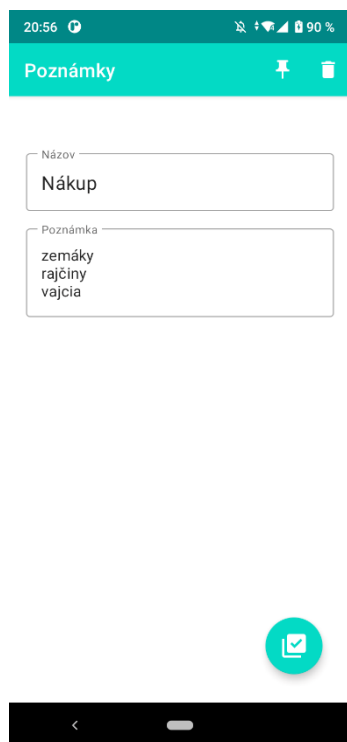
V hornej lište je možné kliknúť na tlačítko lupy, ktorá umožňuje vzhľadávať v poznámkach podľa mena. Toto vyhľadávanie prebieha v reálnom čase a teda není nutné potvrdzovať vyhľadávaný text.



**NoteEditorFragment** – Fragment, ktorý zobrazuje pridávanie/edit poznámky. Umožňuje zadať meno a popis poznámky. V hornej lište je možné pripnúť poznámku tak, aby sa v liste na úvodnej obrazovke zobrazoval hore.

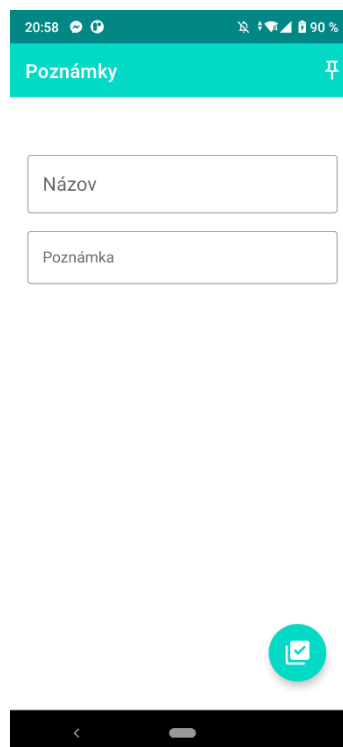
Tento fragment je vyžívaný pri pridávaní nových poznámok a taktiež aj pri editácii existujúcich. Fragment rozlišuje medzi týmito módmí tak, že pri editácii si do fragmentu z **ListOfNotesFragment** pošleme pomocou **SafeArgs** ID editovanej poznámky. Ak takéto ID fragment dostane tak vie, že si ma načítať poznámku z databázy a taktiež povoliť tlačidlo v hornej lište na mazanie poznámky.

Edit existujúcej poznámky



The screenshot shows the 'NoteEditorFragment' in edit mode. At the top, there is a teal header bar with the title 'Poznámky' and two icons: a pushpin and a trash can. Below the header, there are two input fields. The first field is labeled 'Názov' and contains the text 'Nákup'. The second field is labeled 'Poznámka' and contains the text 'zemáky', 'rajčiny', and 'vajcia' on three lines. At the bottom of the screen, there is a floating action button (FAB) with a teal background and a white checkmark icon. The bottom navigation bar is black with a white back arrow icon on the left.

Pridávanie novej poznámky – skryje sa kôš



The screenshot shows the 'NoteEditorFragment' in add mode. At the top, there is a teal header bar with the title 'Poznámky' and one icon: a pushpin. Below the header, there are two input fields. The first field is labeled 'Názov' and is empty. The second field is labeled 'Poznámka' and is empty. At the bottom of the screen, there is a floating action button (FAB) with a teal background and a white checkmark icon. The bottom navigation bar is black with a white back arrow icon on the left.

V aplikácii využívam odporúčanú MVVM architektúru. To znamená, že každý fragment má priradený svoj ViewModel, ktorý udržiava stav v prípade znovu pretvorenia fragmentov. ViewModel využíva **Kotlin Coroutines** na spúšťanie a propagovanie dotazov na databázu aby tieto dotazy nespomaľovali hlavné vykresľovanie vlákno. Tieto coroutines sú viazané na životnosť ViewModelu takže po jeho zničení sa ukončí aj coroutine.

Repository nie je povinná ale odporúčana vrstva v MVVM architektúre. Táto vrstva združuje logiku na jednom mieste. Ak by táto repository vrstva chýbala tak táto logika by bola roztrúsená naprieč všetkými ViewModelmi a jej zmeny by boli ťažšie. V mojom kóde využívam len jednu repository triedu **NoteRepository**, ktorá propaguje volania na databázu.

Model vrstva pracuje s databázou. V aplikácii je jedna tabuľka **Note**, ktorá uchováva jednotlivé poznámky. S databázou sa pracujem pomocou **NoteDao** triedy, ktorá ma v sebe definované všetky potrebné SQL queries.

```
@Entity(tableName = "note")
data class Note(
    val name: String,
    val description: String,
    val pinned: Boolean = false,

    @PrimaryKey(autoGenerate = true)
    val id: Int = 0
)
```



## AndroidX komponenty – Binding

V aplikácii využívam binding, ktorý uľahčuje prácu s views fragmentov. Vďaka tejto knižnici sa jednotlivé layouty vytvoria do tried a s týmito triedami sa pracuje už oveľa ľahšie. Tým pádom sa eliminuje nutnosť hľadať všetky potrebné views pomocou *findViewById*.

```
class ListOfNotesFragment : Fragment() {  
  
    private lateinit var binding: FragmentListOfNotesBinding  
    private val noteViewModel: NoteViewModel by viewModels()  
    private lateinit var searchBar: SearchView  
  
    /**  
     * Povoli horne menu  
     * @param savedInstanceState  
     */  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setHasOptionsMenu(true) //povoli horne menu  
    }  
  
    /**  
     * Inflatne fragment_list_of_notes.xml a po kliknutí na floating action button  
     * sa zobrazí fragment na pridanie novej poznámky  
     * @param inflater  
     * @param container  
     * @param savedInstanceState  
     * @return  
     */  
    override fun onCreateView(  
        inflater: LayoutInflater, container: ViewGroup?,  
        savedInstanceState: Bundle?  
    ): View {  
        binding = FragmentListOfNotesBinding.inflate(inflater)  
        binding.fabAdd.setOnClickListener { it: View! -> {  
            val action =  
                ListOfNotesFragmentDirections.actionListOfNotesFragmentToNoteEditorFragment()  
            it.findNavController().navigate(action)  
        }  
        return binding.root  
    }  
}
```

## LiveData a ViewModel

LiveData je knižnica na pozorovanie zmien (observer pattern), ktorá ma výhodu toho, že je viazaná na životný cyklus komponentu. Vďaka jeho podpore s Android Room stačí pozorovať danú query a prípadne zmeny dát v tej query sa automaticky propagujú pomocou LiveData .

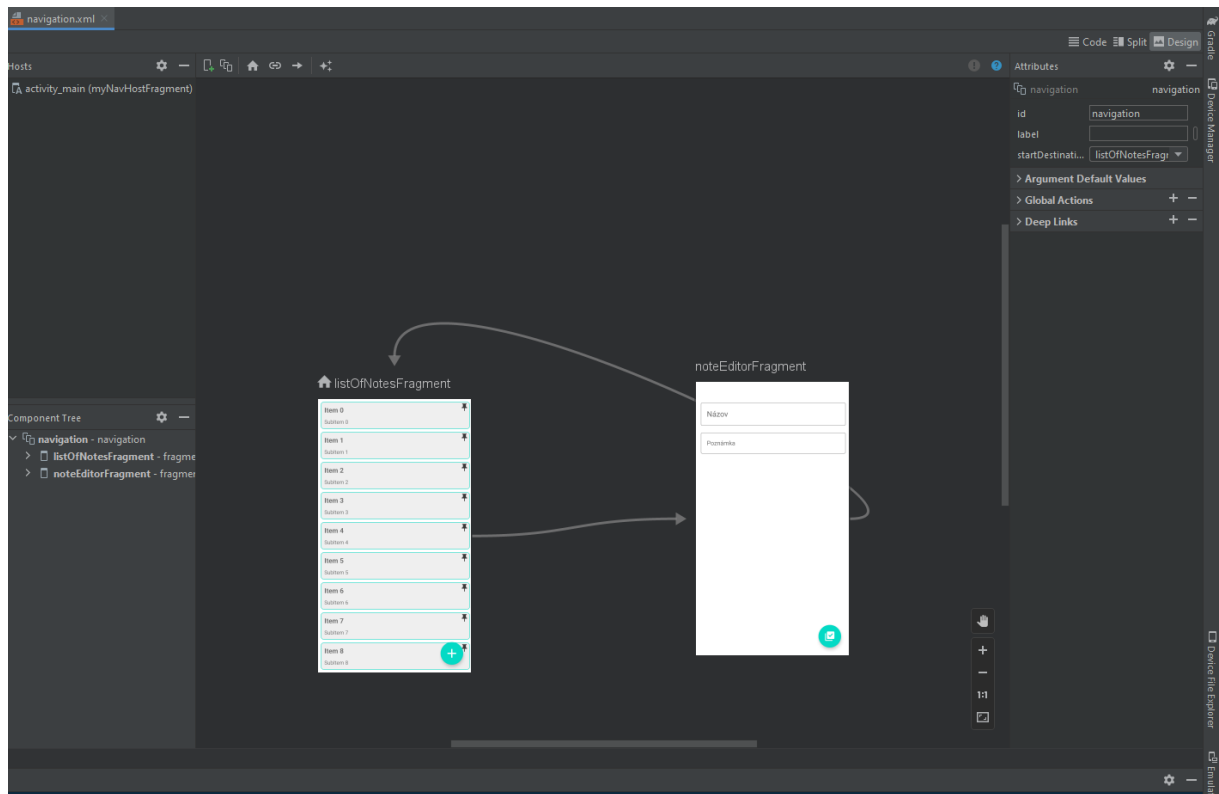
```
class NoteViewModel @Inject constructor(private val repository: NoteRepository) : ViewModel() {
    var allNotesFiltered: LiveData<List<Note>>
    private val filter = MutableLiveData<String>()
    var searchText: String = ""

    init {
        allNotesFiltered = Transformations.switchMap(filter) { filter ->
            repository.searchDatabase(filter).asLiveData()
        }
    }

    /**
     * Metoda ktorá nastaví filter vyhľadavania
     * Ak je vstupný string prázdny tak sa vyhľadava bez filtru
     * Ak je zadán string tak sa vyhľadava podľa zadaneho stringu
     *
     * @param newFilter
     */
    fun setSearchFilter(newFilter: String) {
        val f = when {
            newFilter.isEmpty() -> ""
            else -> "$newFilter"
        }
        filter.postValue(f)
    }
}
```

## Navigation Component a SafeArgs

Tieto 2 knižnice uľahčujú prácu pri prechodoch medzi jednotlivými fragmentami. Stačí v XML definovať prechody a vďaka nim sa v kóde ľahšie vyvolávajú prechody medzi fragmentami. Vďaka **SafeArgs** je jednoduchšie predávanie argumentov medzi týmito fragmentami, napríklad v prípade editovania poznámky keď predávam ID poznámky medzi hlavným fragmentom a fragmentom na editovanie.



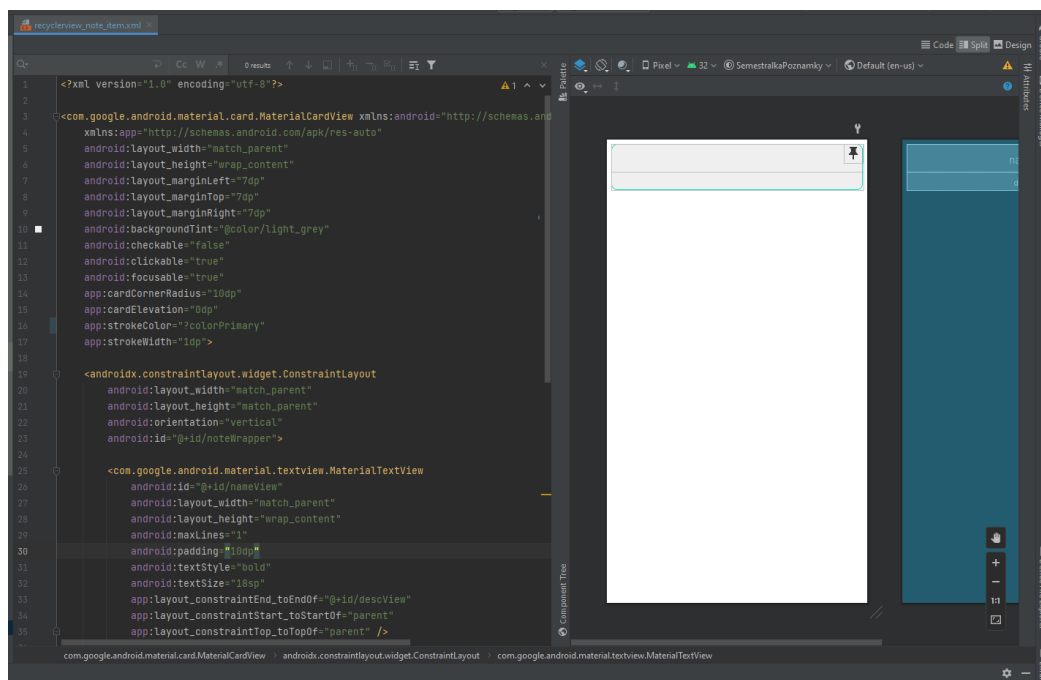
## Android Room Database

Táto knižnica značne uľahčuje prácu s lokálnou databázou.

```
Note.kt x AppModule.kt x NoteRepository.kt x NoteRoomDatabase.kt x NoteDao.kt x
1 package sk.uniza.semestralkapoznamky.data
2
3 import ...
4
5
6 @Entity(tableName = "note")
7 data class Note(
8     val name: String,
9     val description: String,
10    val pinned: Boolean = false,
11
12    @PrimaryKey(autoGenerate = true)
13    val id: Int = 0
14 )
15
16
17 @Database(entities = [Note::class], version = 2, exportSchema = false)
18 abstract class NoteRoomDatabase : RoomDatabase() {
19     abstract fun noteDao(): NoteDao
20 }
```

## Využitie RecyclerView a adaptéru

```
class NoteListAdapter : ListAdapter<Note, NoteListAdapter.NoteViewHolder>(WordsComparator()) {  
  
    /**  
     * Vytvara novu view poznamky  
     *  
     * @param parent  
     * @param viewType  
     * @return  
     */  
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): NoteViewHolder {  
        val itemBinding =  
            RecyclerViewNoteItemBinding.inflate(LayoutInflater.from(parent.context), parent, attachToParent: false)  
        return NoteViewHolder(itemBinding)  
    }  
  
    /**  
     * Nastavuje existujucemu view poznamky dane meno, popis, id a pin  
     *  
     * @param holder  
     * @param position  
     */  
    override fun onBindViewHolder(holder: NoteViewHolder, position: Int) {  
        val current = getItem(position)  
        holder.bind(current.name, current.description, current.id, current.pinned)  
    }  
  
    /**  
     * Pomocna trieda, ktora pracuje s view poznamky a nastavuje jej data  
     *  
     * @property itemBinding  
     */  
    class NoteViewHolder(private val itemBinding: RecyclerViewNoteItemBinding) :  
        RecyclerView.ViewHolder(itemBinding.root) {  
        fun bind(text: String, desc: String, id: Int, pinned: Boolean) {  
            itemBinding.noteWrapper.setOnClickListener { it: View! }  
        }  
    }  
}
```



## Android Dagger Hilt

Externá knižnica ktorá slúži na spravovanie dependencies v kóde. V triede stačí definovať ako vytvoriť inštancie, ktoré potom kompilátor vie injectovať automaticky. Tým pádom nám stačí vytvárať inštancie len na 1 mieste v kóde.

```
14
15 /**
16  * Modul, ktorý ma definované ako vytvárať inštancie jednotlivých tried(dependencies)
17  * Tieto inštancie sú vytvárané ako singleton
18  */
19
20 @Module
21 @InstallIn(SingletonComponent::class)
22 class AppModule {
23
24     @Singleton
25     @Provides
26     fun provideNoteRepository(
27         noteDao: NoteDao
28     ): NoteRepository {
29         return NoteRepository(noteDao)
30     }
31
32     @Singleton
33     @Provides
34     fun provideNoteDao(
35         noteRoomDatabase: NoteRoomDatabase
36     ): NoteDao {
37         return noteRoomDatabase.noteDao()
38     }
39
40     @Singleton
41     @Provides
42     fun provideDatabase(
43         @ApplicationContext app: Context
44     ): NoteRoomDatabase {
45         return Room.databaseBuilder(
46             app,
47             NoteRoomDatabase::class.java,
48             "note_database"
49         ).build()
50     }
51 }
```

```
@HiltViewModel
class NoteEditorModel @Inject constructor(private val repository: NoteRepository) : ViewModel() {
    val pinChecked = MutableLiveData(value: false)
```

## Použitie vlastných tém

```
themes.xml
1 <resources xmlns:tools="http://schemas.android.com/tools">
2   <!-- Base application theme. -->
3   <style name="Theme.SemestraKaPoznamky" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
4     <!-- Primary brand color. -->
5     <item name="colorPrimary">@color/teal_200</item>
6     <item name="colorPrimaryVariant">@color/teal_700</item>
7     <item name="colorOnPrimary">@color/white</item>
8     <!-- Secondary brand color. -->
9     <item name="colorSecondary">@color/teal_200</item>
10    <item name="colorSecondaryVariant">@color/teal_700</item>
11    <item name="colorOnSecondary">@color/black</item>
12    <!-- Status bar color. -->
13    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
14    <!-- Customize your theme here. -->
15  </style>
16 </resources>
```