Žilinská univerzita v Žiline

Fakulta Riadenia a Informatiky

APLIKÁCIA PRE SPRÁVU POZNÁMOK

Semestrálna práca

Filip Sudora

2020/2021

5ZYI23

Obsah

Popis a analýza riešeného problému

Mojim zadaním bolo vypracovať aplikáciu slúžiacu na ukladanie a spracovanie poznámok. V ktorej si používateľ taktiež môže prispôsobovať svoje poznámky a preferencie.

Nakoľko sa na obchodoch s aplikáciami nachádza až neuveriteľné množstvo poznámkových aplikácií, nenašiel som žiadny závažný problém, ktorý by som mohol riešiť pomocou mojej aplikácie.

Najsťahovanejšou poznámkovou aplikáciou podľa Google Play je **Google Keep**, táto aplikácia umožňuje vytvárať poznámky rôznym spôsobom, napr. písaním textu, vložením (odfotografovaním) obrázku, hlasovým zadávaním, kreslením a vytvorením zoznamu nejakých položiek. Jednotlivé poznámky môžeme pripnúť, nastaviť im upozornenie, zdieľať a mnoho ďalšieho. Poznámky si taktiež môžeme archivovať, prípadne zahodiť do koša a opätovne obnoviť.

Ďalšou často sťahovanou aplikáciou je **ColorNote Notepad Notes**. Pomocou tejto aplikácie je možné vytvárať len textové a zoznamové typy poznámok. Avšak oproti Google Keep má používateľské prostredie aj widget podobnejší kancelárskym poznámkam.

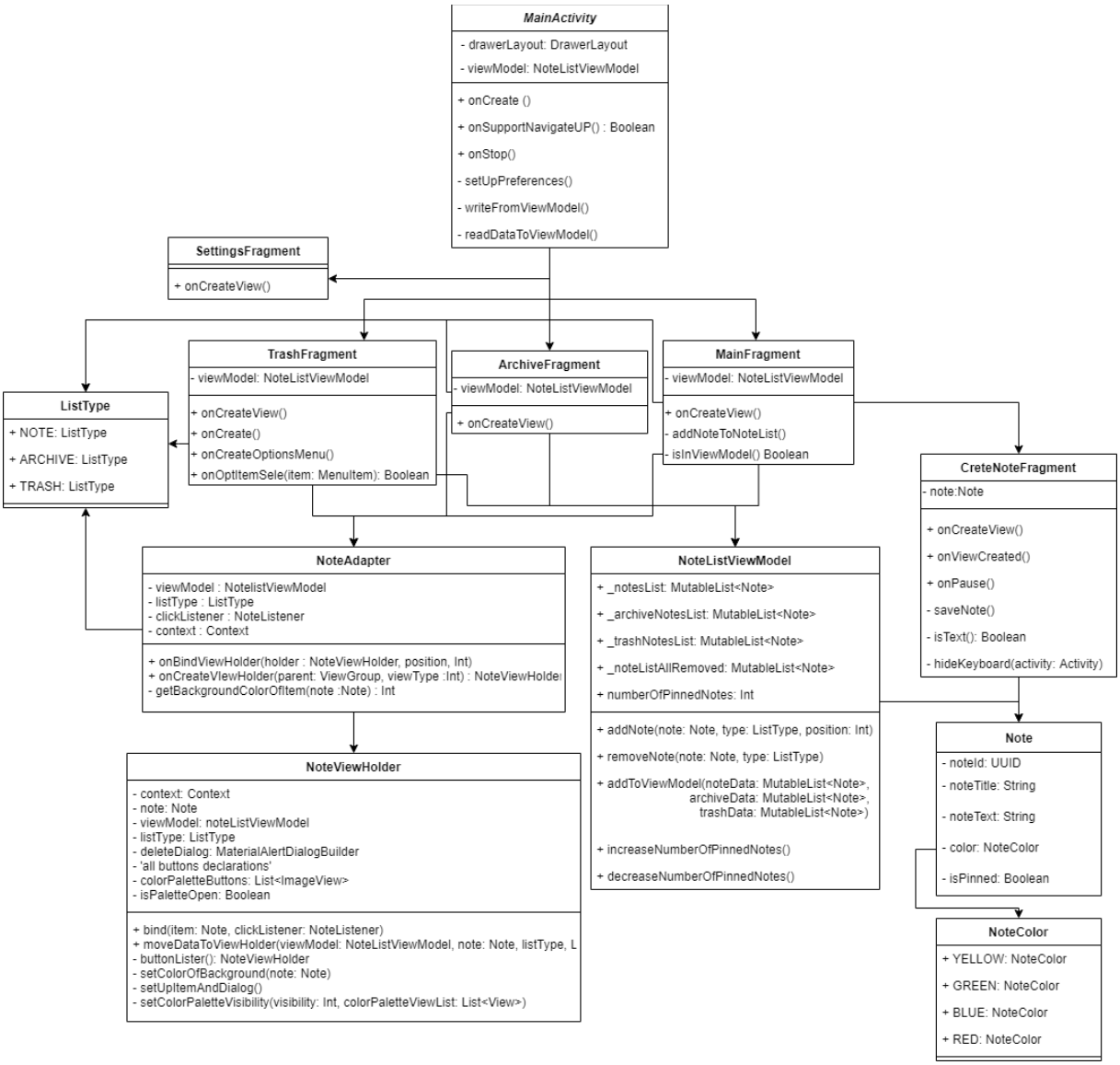
Nakoľko Google má svoju poznámkovú aplikáciu veľmi funkčne aj esteticky spravenú, nedokázal som sa odlíšiť žiadnu funkciu, avšak ak sa zameriam na druhú spomínanú aplikáciu ColorNote, od nej sa líšim jednoduchším používateľským rozhraním rozhraním a väčším priestorom na zobrazovanie vytvorenej poznámky.

Analýza navrhovanej aplikácie

Používateľ bude môcť vytvoriť, upraviť, vymazať poznámky. Taktiež si ich bude môcť archivovať do špeciálneho priečinka, zdieľať, meniť podfarbenie, prípadne nastaviť poznámku ako pripnutú, čím sa mu zobrazí na vrchu zoznamu.

Návrh aplikácie

Po spustení aplikácie sa používateľovi zobrazia všetky jeho uložené poznámky. Tie sa budú zobrazovať na **MainFragment**e. V MainFragmente bude tiež tlačidlo pre vytvorenie novej poznámky, po stačení sa prejde na fragment s názvom **CreateNoteFragment**. Tu používateľ zadá nadpis a text svojej poznámky. Po stlačení tlačidla pre uloženie poznámky sa skontroluje či daná poznámka obsahuje nadpis a text. Ak nie tak používateľa vyzve na zadanie nadpisu/textu, ak áno vytvorí sa nová inštancia dátovej triedy **Note** a poznámka sa uloží do triedy **NoteListViewModel**, ktorá bude v sebe ukladať všetky poznámky. Tie sa následne zobrazia cez RecyclerView v MainFragmente ako miniatúry, pomocou ktorých bude možné zvoliť tlačidlá pre úpravu textu, zmenu podfarbenia, zdieľanie, nastavenie pripnutia a presun poznámky do archívu/koša. Poznámky v archíve/koši budem ukladať do už vytvoreného NoteListViewModelu. Zobrazenie týchto poznámok bude v **ArchiveFragment**e, resp. **TrashFragment**e**.** Používateľ si bude môcť upraviť svoje preferencie pomocou **SettingsFragment**u, kde si bude môcť nastaviť zapnutie tmavého režimu alebo zobrazovanie nových poznámok na vrchu zoznamu.



Obrázok 1 UML diagram tried

Popis implementácie

Aktivity a fragmenty

MainActivity

Jediná a hlavná aktivita aplikácie, nastavuje navigation drawer, preferencie používateľa pred spustením aplikácie, načítava a zapisuje poznámky do súboru.

MainFragment

Fragment, ktorý sa zobrazuje po spustení aplikácie. Zobrazuje uložené poznámky a pridáva nové poznámky do zoznamu noteList v NoteListViewModely. Zavolá NoteAdapter, cez ktorý sa tieto poznámky zobrazujú. Keďže zoznam poznámok je LiveData môžem ich observovať a keď zistím ich zmenu oboznámim o tom adaptér.

CreateNoteFragment

Fragment kde používateľ zadáva text, ktorý sa neskôr uloží ako poznámka. Fragment zobrazoje dva edittexty, prvý slúži na zadanie nadpisu poznámky, druhý na text poznámky. Stlačením klávesy s ikonou „fajka“ sa poznámka uloží ak je v nadpise aj texte nejaký text. Po stlačení klávesy „späť“ sa poznámka neuloží.

ArchiveFragment

V tomto fragmente sa ukladajú archivované poznámky. Z tohto fragmentu môžeme poznámku presunúť do koša alebo do hlavných poznámok. Prípadne zdieľať.

TrashFragment

V tomto fragmente sú uložené poznámky na zahodenie. Z tohto fragmentu môžeme poznámku presunúť do hlavných poznámok, alebo poznámku natrvalo vymazať. Pred úplným vymazaním poznámky sa aplikácia pre istotu spýta, či tak chce skutočne učiniť.

SettingsFragment

Fragment, v ktorom si používateľ môže nastaviť svoje preferencie. Dá sa nastaviť zapnutie/vypnutie tmavého režimu alebo pridávanie nových poznámok na začiatok hlavného zoznamu alebo na koniec.

Využitie AndroidX komponentov

Data Binding

Data Binding som používal pre jednoduchší prístup ku komponentom layoutu. Ako príklad môžem uviesť časť triedy CreateNoteFragment, kde si nastavujem binding a následne ho využívam pre nastavenie hodnôt do edittextov(pri upravovaní poznímky), ale aj pre prístup k tlačidlám vo fragmente.

```
class CreateNoteFragment : Fragment() {
    private lateinit var binding: FragmentCreateNoteBinding
    private val args by navArgs<CreateNoteFragmentArgs>()
    private var note = Note(UUID.randomUUID(), noteTitle: "", noteText: "", NoteColor.YELLOW)

    override fun onCreateView(
            inflater: LayoutInflater, container: ViewGroup?,
            savedInstanceState: Bundle?
    ): View {
        //skrytie action baru pred vstupom do fragmentu
        (activity as AppCompatActivity?)!!.supportActionBar!!.hide()

        binding = DataBindingUtil.inflate(inflater, R.layout.fragment_create_note, container, attachToParent: false)
        try {
            note = args.editNote!!
        } catch (e: NullPointerException) {}
        //napise do editovacich poli titulok a text poznamky
        binding.noteTitleInput.setText(note.noteTitle, TextView.BufferType.EDITABLE)
        binding.noteInput.setText(note.noteText, TextView.BufferType.EDITABLE)

        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        //ak je stacene tlacidlo spat, chod na main fragment
        binding.backButton.setOnClickListener { view: View ->
            view.findNavController().navigate(CreateNoteFragmentDirections.actionCreateNoteFragmentToMainFragment(note))
        }
        //ak je stlacene tlacitlo ulozenia poznamky, skontroluj ci je v poznamke text, uloz poznamku a chod na main fragment
        binding.doneButton.setOnClickListener {  it: View!
            if (isText()) {
                saveNote()
                view.findNavController().navigate(CreateNoteFragmentDirections.actionCreateNoteFragmentToMainFragment(note))
            }
        }
    }
```

Obrázok 2 ukážka data bindingu v triede CreateNoteFragment

LifeCycles

Okrem onCrete() metódy používam LifeCycles hlavne pri odídení z fragmentu/ukončení aktivity. Ako napríklad pri spustení aplikácie si nastavím view model, binding, načítam uložené poznámky, nastavím navigation drawer a nastavím preferencie používateľa. Ďalej pri ukončení aplikácie si uložím poznámky alebo pri pozastavení fragmentu CreateNoteFragment skryjem klávesnicu.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val binding = DataBindingUtil.setContentView<ActivityMainBinding>( activity: this, R.layout.activity_main)
    drawerLayout = binding.drawerLayout
    viewModel = ViewModelProvider( owner: this).get(NoteListViewModel::class.java)
    //nacita ulozene poznamky do view modelu
    readDataToViewModel()
    //nastav navigation drawer a kontroler
    val navController = this.findNavController(R.id.myNavHostFragment)
    NavigationUI.setupActionBarWithNavController( activity: this, navController, drawerLayout)
    NavigationUI.setupWithNavController(binding.navView, navController)
    //nastav preferencie
    setUpPreferences()
}
```

Obrázok 3 metóda onCreate() hlavnej aktivity

```
/**
 * Pri konci aplikacie uloz view model
 */
override fun onStop() {
    super.onStop()
    writeFromViewModel()
}
```

Obrázok 4 metóda onStop() hlavnej aktivity

```
/**
 * Pri odchode z fragmentu skry klavesnicu
 */
override fun onPause() {
    super.onPause()
    hideKeyboard(this.requireActivity())
}
```

Obrázok 5 metóda onPause() triedy CreateNoteFragment

LiveData

LiveData používam vo view modely keďže potrebujem pozorovať zmeny v zoznamoch, keď sa poznámka vymaže/pridá/upraví.

```
private var _notesList = MutableLiveData<MutableList<Note?>>().apply { this.value = __notesList }
val noteList : LiveData<MutableList<Note?>>
    get() = _notesList

private var _archiveList = MutableLiveData<MutableList<Note?>>().apply { this.value = __archiveList }
val archiveList : LiveData<MutableList<Note?>>
    get() = _archiveList

private var _trashList = MutableLiveData<MutableList<Note?>>().apply { this.value = __trashList }
val trashList : LiveData<MutableList<Note?>>
    get() = _trashList
```

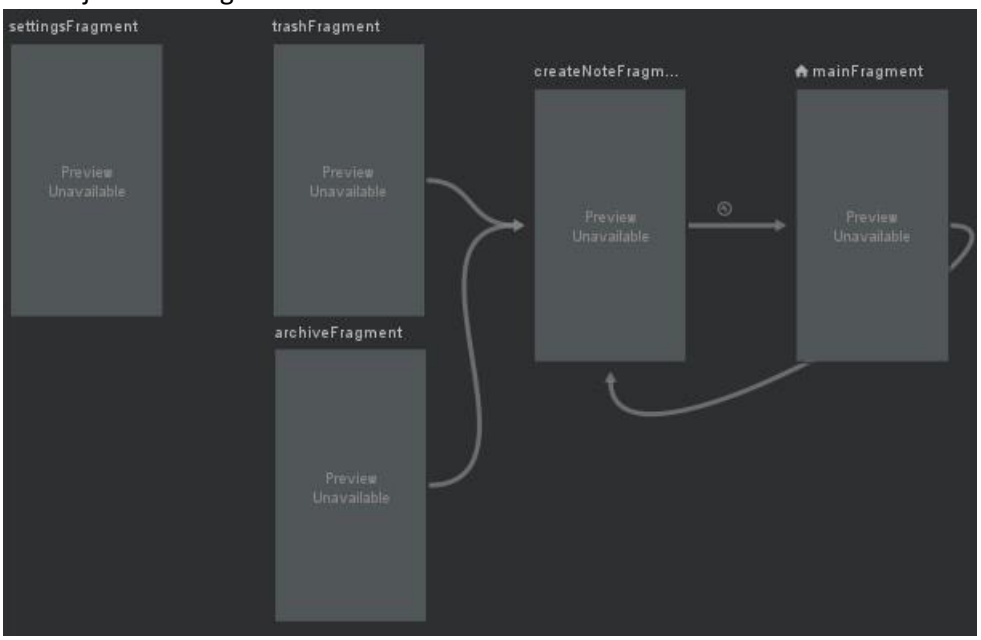
Obrázok 6 deklarácia zoznamov zaobalených v LiveData

```
//observer zmeny zoznamu poznamok
viewModel.noteList.observe(viewLifecycleOwner, Observer {  it: MutableList<Note?>!
    it?.let {  it: MutableList<Note?>
        adapter.submitList(it)
    }
    adapter.notifyDataSetChanged()
    if (it.size > 0) {
        binding.textViewNote.text = ""
    } else {
        binding.textViewNote.text = "NO NOTES TO DISPLAY"
    }
})
```

Obrázok 7 využitie LiveData v MainFragmente pri zistení zmeny v zozname oboznám o tom adaptér

Navigation

Navigačný komponent používam pri prechodoch z jedného fragmentu do druhého, či už po stlačení tlačidla na fragmente alebo pri navigation drawere. Taktiež pomocou Safe Args posúvam dáta z jedného fragmentu do druhého.



Obrázok 8 ukážka navigačných ciest

```
override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)
    //ak je stacene tlacidlo spat, chod na main fragment
    binding.backButton.setOnClickListener { view: View ->
        view.findNavController().navigate(CreateNoteFragmentDirections.actionCreateNoteFragmentToMainFragment(note))
    }
    //ak je stlacene tlacitlo ulozenia poznamky, skontroluj ci je v poznamke text, uloz poznamku a chod na main fragment
    binding.doneButton.setOnClickListener {  it: View!
        if (isText()) {
            saveNote()
            view.findNavController().navigate(CreateNoteFragmentDirections.actionCreateNoteFragmentToMainFragment(note))
        }
    }
}
```

Obrázok 9 príklad navigačných ciest v CreateNoteFragment

ViewModel

ViewModel používam pre uchovávanie zoznamov poznámok a manipuláciu s nimi (presuny/mazania), ale aj aby sa mi pri otočení displeja poznámky zachovali správne.

```
private var __notesList = mutableListOf<Note?>()
private var __archiveList = mutableListOf<Note?>()
private var __trashList = mutableListOf<Note?>()
```

Obrázok 10 deklarácia zoznamov poznámok vo view modely

```
/**
 * Prida poznamku do zadaneho zoznamu na zadane miesto
 *
 * @param note poznamka na pridanie
 * @param type typ zoznamu kam sa ma poznamka pridat
 * @param position na aku poziciu sa ma poznamka pridat
 */
fun addNote(note: Note, type: ListType, position: Int = -1) {
    when(type) {
        ListType.NOTE ->{
            //ak sa nenastavi hodnota position prida na koniec zoznamu
            if (position == -1) {
                __notesList.add(note)
            } else {
                __notesList.add(position, note)
            }
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _notesList.postValue(__notesList)
        }
        ListType.ARCHIVE ->{
            __archiveList.add(note)
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _archiveList.postValue(__archiveList)
        }
        ListType.TRASH ->{
            __trashList.add(note)
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _trashList.postValue(__trashList)
        }
    }
}
```

Obrázok 11 pridanie poznámky do určeného zoznamu

```
/**
 * Odstrani zadanu poznamku zo zadaneho zoznamu
 *
 * @param note poznamka na odstranenie
 * @param type typ zoznamu, z ktoreho sa odstrani poznamka
 */
fun removeNote(note: Note, type: ListType) {
    when(type) {
        ListType.NOTE ->{
            //odstrani poznamku zo zoznamu
            __notesList.remove(note)
            //je potrebne si zaznamenavat aj poznamky, ktore boli odstranene zo zoznamu
            //kedze v safe args zostava hodnota z CreateNoeFragment, poznamky by sa inak duplikovali
            __noteListAllRemoved.add(note)
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _notesList.postValue(__notesList)
        }
        ListType.ARCHIVE ->{
            __archiveList.remove(note)
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _archiveList.postValue(__archiveList)
        }
        ListType.TRASH ->{
            __trashList.remove(note)
            //nastavi zoznam poznamok z vedlajsieho vlakna
            _trashList.postValue(__trashList)
        }
    }
}
```

Obrázok 12 odobratie poznámky z určeného zoznamu

Použitie externej knižnice

Material Design

Do aplikácie som z tejto knižnice použil komponenty ako „Text fields“, „Switches“ a „Dialogs“. Text fields som použil pri vytváraní/upravovaní poznámky, Switches vo fragmente s nastaveniami a Dialogy pri trvalom odstraňovaní poznámky/poznámiek z koša.

```
//zobrazi dialog ci skutocne chce pouzivatel vymazat vsetky poznamky
val deleteDialog : MaterialAlertDialogBuilder = MaterialAlertDialogBuilder(requireContext())
        .setTitle("Delete note")
        .setMessage("Are you sure you want to delete all notes?")
        .setNegativeButton("No")
        { _, _ -> }
deleteDialog.setPositiveButton("Yes")
{ _, _ ->
    //vymaze vsetky poznamky z kosa
    //a upozorni adapter
    viewModel.trashList.value?.clear()
    adapter.notifyDataSetChanged()
    binding.textViewTrash.text = "YOUR TRASH IS EMPTY"
}.create().show()
```

Obrázok 13 vytvorenie a zobrazenie Dialogu pomocou externej knižnice