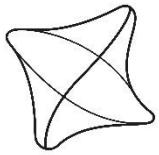


ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Vrecková knižnica

VAMZ Semestrálna práca

Jan Drahos, 5ZSY33

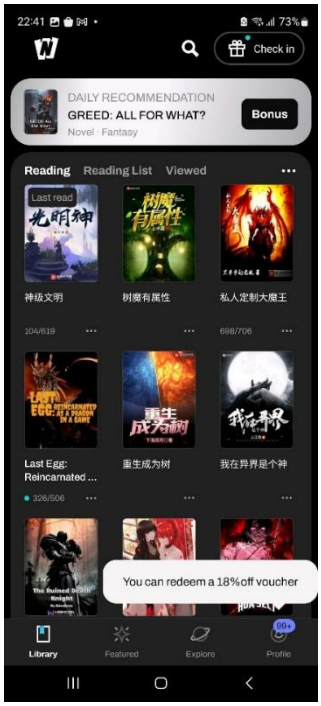
Ciel práce

Cieľom práce je vytvorenie Android aplikácie pomocou JetpackCompose a jazyka Kotlin. Táto aplikácia má umožňovať užívateľom lokálne agregovať knihy z online zdrojov do prenosnej knižnice. Táto knižnica by mala umožňovať vyhľadávanie kníh, pridávanie nových kníh, mazanie starých a ich triedenie na základe ich stavu (plánujem čítať, prečítal som, čítam teraz).

Podobné aplikácie

Webnovel

Webnovel je globálne prístupná platforma pre publikovanie a čítanie kníh s viac ako 10 miliónmi stiahnutí a 351 tisíc hodnotení na google store.



Ponuka funkcie pre čitateľov ako aj autorov, kde čitatelia môžu komentovať, písať recenzie, a pridávať knihy do knižnice a čítať knižky.

Takisto dokáže odporučiť užívateľom nové knihy na čítanie na základe ich preferencii a aktuálne populárnych kníh.

Takisto podporuje systém odmien ktorý dovoľuje čitateľom čítať limitovaný počet kapitol denne za účasť pri hlasovaniach ako aj sociálne interagovanie v komunite.

Na základe týchto motivačných odmien vytvárajú vlastné priečky pre knihy na základe obdŕžaných hlasov, populárnosti, ako aj angažovanosti jéj čitateľov.

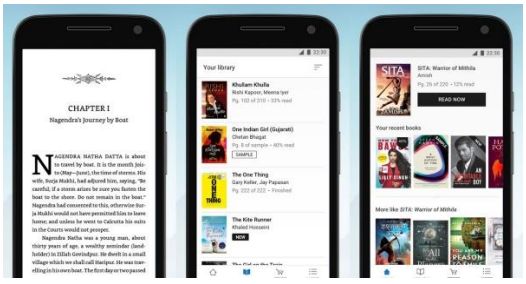
MegaNovel

MegaNovel, rovnako ako Webnovel je aplikácia ktorá dovoľuje užívateľom čítať a hodnotiť knihy, ako ich aj pridávať do knižníc. Avšak je omnoho menej populárna ako Webnovel, len 500 tisíc stiahnutí na google store.

Amazon Kindle

Platforma pre čítanie kníh od spoločnosti Amazon ktorá má aj vlastnú aplikáciu pre mobilné zariadenia. Je omnoho populárnejšia ako Webnovel a MegaNovel s viac ako 100 miliónov stiahnutí na google store.

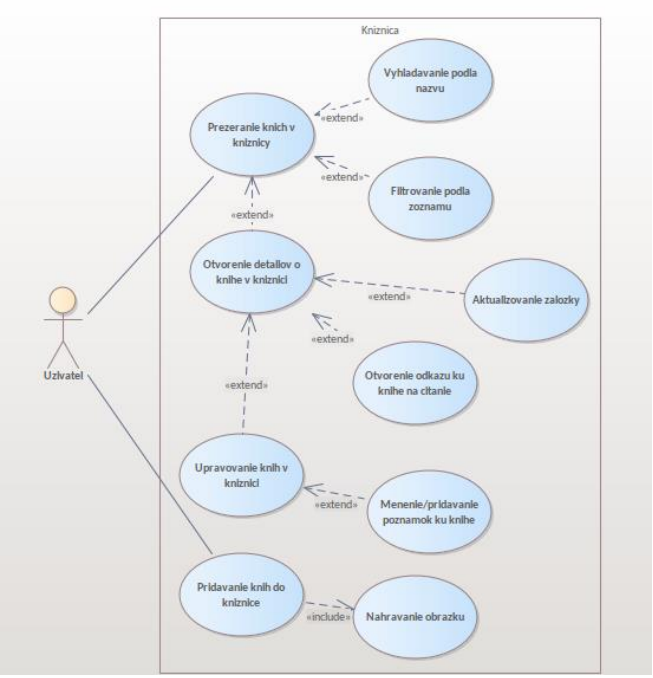
Zároveň od Webnovel a MegaNovel poskytuje okrem webnoviel aj profesionálne vydávané knihy, ale jéj sociálne funkcie sú omnoho menej rozvinuté.



Rozdiely mojej aplikácii oproti existujúcim aplikáciám

Moja aplikácia sa odlišuje od vyššie spomenutých aplikácii tým že priamo nehostuje knihy, ale namiesto toho ponuka možnosť vytvárať knižnice ktoré agregujú knihy s viacerých zdrojov pomocou užívateľom vložených web linkov.

Návrh aplikácie

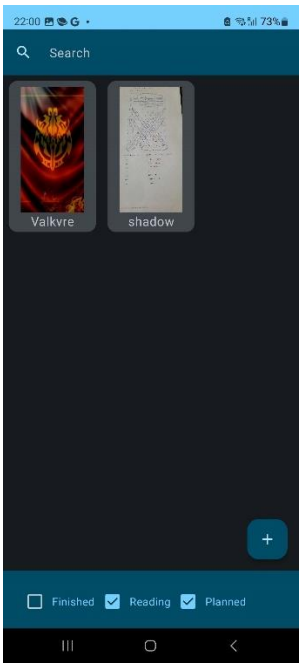


Obrazovka: Library

Ústrednou funkciou mojej aplikácie je agregovanie online kníh s viacerých zdrojov. Za týmto účelom obsahuje obrazovku knižnica v ktorej užívateľ môže filtrovať, vyhľadávať, a prezerať pridané knihy.

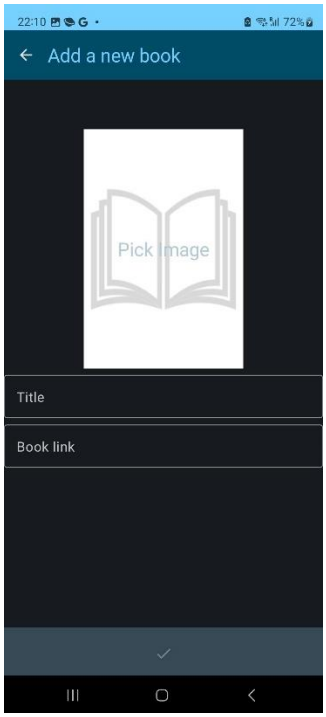
Do tejto knižnice užívateľ môže pridávať knihy prostredníctvom Book Entry obrazovky ktorý sa otvára tlačidlom + v pravom dolnom rohu obrazovky knižnice.

Užívateľ môže takisto v knižnici otvoriť detaily o danej knihe kliknutím na ňu.



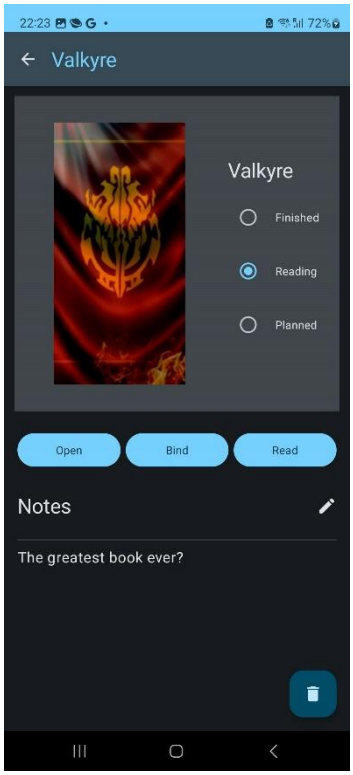
Obrazovka: Book Entry

Obrazovka obsahuje formulár na pridávanie nových kníh do knižnice. Užívateľ vyberá obrázok pre knihu, zadáva jej názov a odkaz na stránku/aplikáciu, v ktorej ju môže prečítať. Vstup užívateľa je overovaný a verifikovaný.



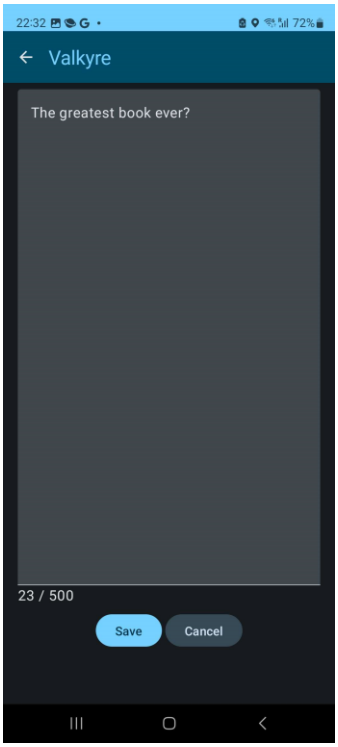
Obrazovka: Book Details

Obrazovka zobrazuje detailné informácie o knihe, ako sú poznámky. Tiež umožňuje operácie nad danou knihou, ako je jej otvorenie pomocou jej odkazu, jej vymazanie, presmerovanie do nástroja na úpravu poznámok, napojenie knihy na widget, ako aj otvorenie knihy vo WebView s funkciou vytvárania záložiek.



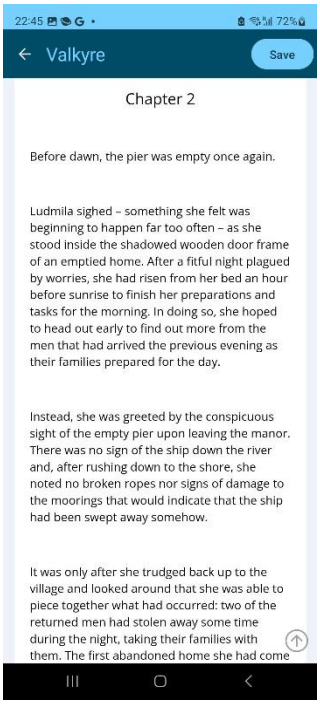
Obrazovka: Book Notes Editor

Obrazovka umožňuje užívateľovi písať krátke poznámky k vybranej knihe. Obsahuje vstupné pole na písanie, počítadlo ukazujúce počet znakov a dve tlačidlá na uloženie zmien alebo ich zahodenie. Obidve tlačidlá vrátia užívateľa späť na obrazovku Book Details.



Obrazovka: Book Reader

Obrazovka pomocou WebView zobrazí danú knihu na kapitole určenej záložkou. Umožňuje ukladanie aktuálnej strany a kapitoly pre knihu pomocou tlačidla na hornej lište. Nemusí fungovať vo všetkých prípadoch kvôli DRM ochranám niektorých stránok s knihami.



Widget: Quick Access

Miniaplikácia widget, ktorá umožňuje užívateľom prepojiť zvolenú knihu na widget, ktorý je následne možné umiestniť na plochu pre rýchly prístup k knihe kliknutím na neho.

Implementácia aplikácie

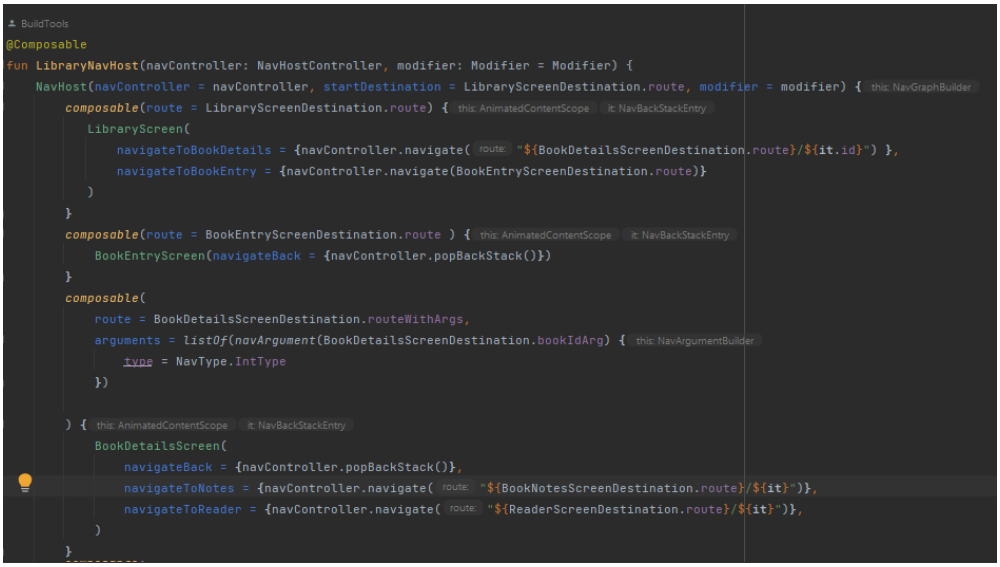
Implementácia navigácie

Navigácia medzi jednotlivými obrazovkami je riešená pomocou NavHostController. Pričom pre každú obrazovku je zadefinovaná cesta pomocou rozhrania NavigationInterface.

Definícia cesty do BookEntry obrazovky:

```
BuildTools
object BookEntryScreenDestination : NavigationDestination {
    override val route: String = "bookEntry"

}
```

Následne sú tieto cesty používané na zadefinovanie navigačného grafu medzi obrazovkami v súbore NavigationGraph.



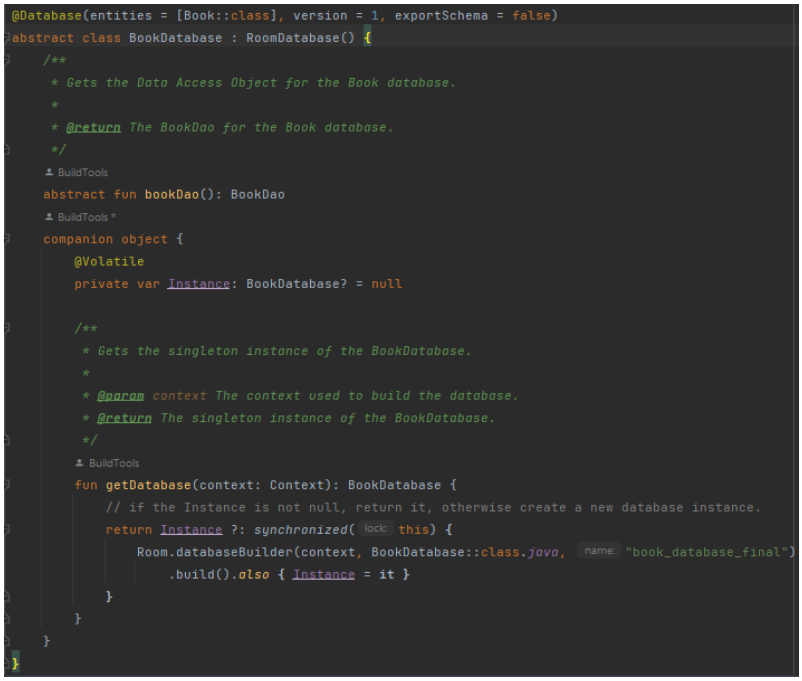
```
BuildTools
@Composable
fun LibraryNavHost(navController: NavHostController, modifier: Modifier = Modifier) {
    NavHost(navController = navController, startDestination = LibraryScreenDestination.route, modifier = modifier) { this: NavGraphBuilder
        composable(route = LibraryScreenDestination.route) { this: AnimatedContentScope  it: NavBackStackEntry
            LibraryScreen(
                navigateToBookDetails = {navController.navigate( route: "${BookDetailsScreenDestination.route}/${it.id}") },
                navigateToBookEntry = {navController.navigate(BookEntryScreenDestination.route)}
            )
        }
        composable(route = BookEntryScreenDestination.route ) { this: AnimatedContentScope  it: NavBackStackEntry
            BookEntryScreen(navigateBack = {navController.popBackStack()})
        }
        composable(
            route = BookDetailsScreenDestination.routeWithArgs,
            arguments = listOf(navArgument(BookDetailsScreenDestination.bookIdArg) { this: NavArgumentBuilder
                type = NavType.IntType
            })

        ) { this: AnimatedContentScope  it: NavBackStackEntry
            BookDetailsScreen(
                navigateBack = {navController.popBackStack()},
                navigateToNotes = {navController.navigate( route: "${BookNotesScreenDestination.route}/${it}")},
                navigateToReader = {navController.navigate( route: "${ReaderScreenDestination.route}/${it}")},
            )
        }
```

Implementácia databázy

Databáza je implementovaná pomocou technológie Room. Pričom má zadefinované DAO.

```
@Dao
interface BookDao {
    BuildTools
    @Insert(onConflict = OnConflictStrategy.IGNORE)
    suspend fun insert(book: Book) : Long

    BuildTools
    @Update
    suspend fun update(book: Book)

    BuildTools
    @Delete
    suspend fun delete(book: Book)

    new *
    @Query("SELECT * from books WHERE id = :id")
    fun getBook(id: Int): Flow<Book>

    new *
    @Query("SELECT * from books ORDER BY title ASC")
    fun getAllBooks(): Flow<List<Book>>
}
```

Jeho databázu:



```
@Database(entities = [Book::class], version = 1, exportSchema = false)
abstract class BookDatabase : RoomDatabase() {
    /**
     * Gets the Data Access Object for the Book database.
     *
     * @return The BookDao for the Book database.
     */
    BuildTools
    abstract fun bookDao(): BookDao
    BuildTools *
    companion object {
        @Volatile
        private var Instance: BookDatabase? = null

        /**
         * Gets the singleton instance of the BookDatabase.
         *
         * @param context The context used to build the database.
         * @return The singleton instance of the BookDatabase.
         */
        BuildTools
        fun getDatabase(context: Context): BookDatabase {
            // if the Instance is not null, return it, otherwise create a new database instance.
            return Instance ?: synchronized( lock: this) {
                Room.databaseBuilder(context, BookDatabase::class.java, name: "book_database_final")
                    .build().also { Instance = it }
            }
        }
    }
}
```

Ukladanie obrázkov

Každá kniha v aplikácia ma vlastný príbalový obrázok. Tieto nie sú priamo uložené v databáze, ale len cesta k ním.

Každá kniha v aplikácii má vlastný príbalový obrázok, ktorý nie je priamo uložený v databáze, ale len jeho cesta. Photo Picker umožňuje bezpečný výber obrázkov z galérie.

Získanie obrázka s Photo Picker:

```
val launcher = rememberLauncherForActivityResult(
    contract = ActivityResultContracts.GetContent()
) { uri: Uri? ->
    uri?.let { onValueChange(bookTitle, bookLink, it) }
}
```

Takto získané obrázky sú následne kopírované do interného úložiska aplikácie, aby sa predišlo problémom s trvalým uchovaním oprávnení na prístup k obrázku alebo zásahom do obrázka zo strany inej aplikácie.

Uloženie obrázka do interného úložiska aplikácie:

```
/**
 * Saves the image locally and returns the new URI.
 *
 * @param uri The original URI of the image.
 * @param id The ID of the book.
 * @return The new URI of the saved image.
 */
BuildTools *
private suspend fun saveImageLocally(uri: Uri, id: Long): Uri = withContext(Dispatchers.IO) { this: CoroutineScope
    val source = ImageDecoder.createSource(context.contentResolver, uri)
    val bitmap = Bitmap.createScaledBitmap(ImageDecoder.decodeBitmap(source),  dstWidth: 150,  dstHeight: 300,  filter: true)

    val filename = "${id}.jpg"
    val file = File(context.filesDir, filename)
    FileOutputStream(file).use { out ->
        bitmap.compress(Bitmap.CompressFormat.JPEG,  quality: 100, out)
        out.flush()
    }

    Uri.fromFile(file) ^withContext
}
```

Oddelenie logiky od užívateľského rozhrania

ViewModel

Logika je oddelená pomocou ViewModel, ktorý zabezpečuje aj získavanie dát pre inicializovanie daných obrazoviek. Pre každú obrazovku je definovaný vlastný ViewModel.

Ukážka ViewModel pre obrazovku BookEntry:

```
BuildTools
class BookEntryViewModel(private val bookRepository: BookRepository, private val context: Context) : ViewModel() {

    BuildTools
    var bookEntryUIState by mutableStateOf(BookEntryUIState())
        private set

    BuildTools
    fun updateState(title : String, link : String, imageUri : Uri) {
        bookEntryUIState = BookEntryUIState(title, link, imageUri)
    }

    BuildTools
    fun validateInput(): Boolean {

        // Check if the title, link, and imageUri are not empty
        if (bookEntryUIState.title.isBlank() ||
```

Všetky ViewModely su vytvárane pomocou centrálne ViewProvider.

UiState

Zároveň sú dočasné dáta potrebné pre vykreslenie obrazovky oddelene do UIState, pričom každá obrazovka má vlastný UIState.

Ukážka UIState pre obrazovku BookEntry:

```kotlin
data class BookEntryUIState(
    var title: String = "",
    var link: String = "",
    var imageUri: Uri = Uri.EMPTY
) {
    /**
     * Converts the UI state to a Book object.
     *
     * @return A Book object with the current state values.
     */
    BuildTools *
    fun getBook(): Book {
        return Book(
            title = title,
            link = link,
            imageUri = imageUri.toString(),
            status = 1,
            notes = "",
            bookMarkUrl = link
        )
    }
}
```

Jediný zdroj pravdy

Aplikácia zachováva princíp jediného zdroja pravdy tak, že trvalé informácie sú získavané z Room databázy, a dočasné sú uložené v jednotlivých UIState, ktoré získavajú dáta z Room databázy a užívateľských vstupov. Jednotlivé prvky užívateľského rozhrania získavajú svoje dáta od príslušného UIState smerom odhora. Zároveň všetok užívateľský vstup je posielaný smerom nahor do UIState a ViewModel.

Implementácia Widget

Toto je implementované pomocou technológie Glance, ktorá umožňuje využívanie JetpackCompose na vytváranie widgetov.

```kotlin
BuildTools *
class QuickAccessWidget : GlanceAppWidget() {

    /**
     * Provides the content for the Glance widget.
     *
     * @param context The context to access resources.
     * @param id The GlanceId of the widget.
     */
    BuildTools *
    override suspend fun provideGlance(context: Context, id: GlanceId) {
        val sharedPreferences = context.getSharedPreferences( name: "widget_prefs", Context.MODE_PRIVATE)
        val bookId = sharedPreferences.getInt( key: "book_id",  defValue: -1)

        val book = if (bookId != -1) getBookById(bookId, context) else null

        provideContent {
            book?.let { it: Book
                QuickAccessWidgetLayout(book = book, context = context)
            }
        }
    }
```

```xml
<appwidget-provider
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:minWidth="65dp"
    android:minHeight="65dp"
    android:updatePeriodMillis="86400000"
    android:widgetCategory="home_screen"
    android:initialLayout="@xml/quick_access_widget_info">
</appwidget-provider>
```

Implementácia obrazoviek

V aplikácia je implementovaných celkovo 5 obrazoviek, pričom pre každú je zadefinovane jej užívateľské rozhranie pomocou JetpackCompose, dáta pomocou UIState a logika pomocou ViewModel.