ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia
a informatiky

Semestrálna práca z predmetu

vývoj aplikácií pre mobilné zariadenia

VOCABRY

Vypracoval: Filip Ďurana

Študijná skupina: 5ZYR21

Akademický rok: 2025

V Žiline dňa 3.6

Obsah

Obrázky

Úvod

V tejto semestrálnej práci predstavujeme mobilnú aplikáciu s názvom Vocabry, ktorej hlavným cieľom je podpora a rozvoj slovnej zásoby v cudzom jazyku. Inšpiráciu na jej vytvorenie sme čerpali z bežného života, kde čoraz rýchlejšie tempo doby zvyšuje nároky na ovládanie cudzích jazykov. Efektívna komunikácia je nevyhnutná časť osobného aj pracovného života , a preto znalosť slovnej zásoby tvorí základ jazykového vzdelávania.

Cieľom tejto práce ja navrhnúť a vytvoriť funkčnú aplikáciu, ktorá používateľom poskytne jednoduchý a prístupný nástroj na systematické učenie sa nových slov. Aplikácia taktiež odráža skúsenosti, ktoré sme nadobudli počas 13 týždňov štúdia.

Krátka analýza

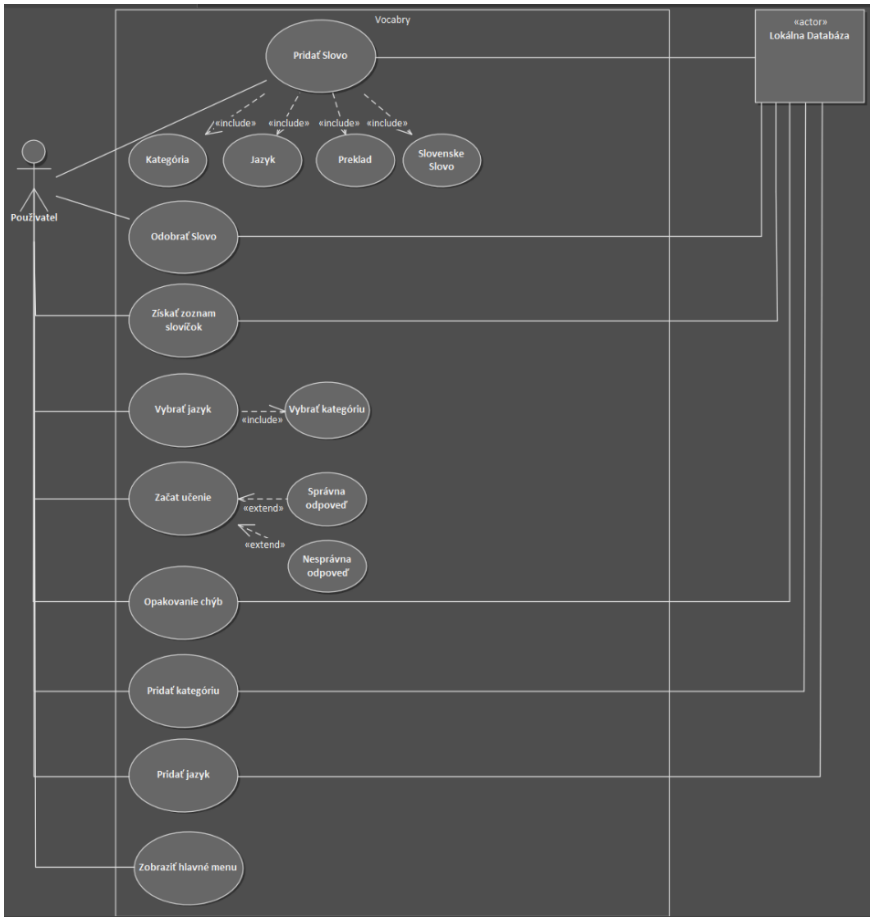
Naša aplikácia je určená pre bežných používateľov, ktorí majú chuť vzdelávať sa a posúvať svoju slovnú zásobu na vyššiu úroveň. Po spustení sa používateľ dostane do hlavného menu, kde si môže vybrať čo chce robiť a či sa chce ihneď začať vzdelávať, alebo najskôr pridať vlastné slovíčka.

Ak si zvolí možnosť „Štart“, aplikácia ho presmeruje na výber jazyka, v ktorom sa chce učiť. Po výbere jazyka pokračuje na obrazovku výberu kategórie (napr. cestovanie, jedlo, technológie atď.).

Po výbere kategórie sa spustí vzdelávací režim. Na obrazovke sa zobrazí otázka s viacerými možnosťami odpovede. Ak používateľ vyberie správnu odpoveď, tlačidlo sa zafarbí na zeleno, započíta sa správna odpoveď a pokračuje sa ďalej. Ak však zvolí nesprávnu možnosť, dané slovíčko sa automaticky uloží do kategórie **„Chyby“**, kde si ich môže neskôr zopakovať a zlepšiť tak svoje znalosti.

Aplikácia obsahuje rôzne preddefinované kategórie a jazyky, ktoré sú plne upraviteľné. Používateľ si navyše môže vytvoriť vlastné kategórie a jazyky podľa vlastných potrieb.

Ak v hlavnom menu zvolí možnosť **„Words“**, bude presmerovaný na obrazovku, kde môže pridať nové slovíčko s prekladom do zvolenej kategórie a jazyka. Ak chce vytvoriť nový jazyk alebo kategóriu, stačí napísať požadovaný názov a systém ich automaticky vytvorí a následne umožní pridávať nové slovíčka.



Obrázok 1. UseCase diagram

Návrh riešenia

V tejto časti si povieme niečo bližšie o Clean architektúre našej práce, ukážeme si vzťahy medzi jednotlivými triedami a do ktorej vrstvy patria.

Clean architektúra je vzor navrhovania softvérových systémov, ktorý odďeluje jednotlivé vrstvy aplikácie podľa ich zodpovednosti. Cieľom Clean architektúry je udržatelnosť, rozšíritelnosť, testovateľnosť jednotlivých častí a nezávislosť na frameworkoch, databázach či UI. Pri vývoji android aplikácii sa Clean achitektúra rozďeľuje na 3 vrstvy:

1. **Prezentačná (UI)** – zobrazenie dát a spracovanie interakcií používateľa
2. **Doménová vrstva** - bussines logika aplikácie, obsahuje useCasy a rozohrania
3. **Dátová vrstva** – implementácia repositárov, prístup k databáze alebo sieti

UI nevie nič o implementácii databázy a usacasy nevedia ako sa presne získavajú alebo ukladajú dáta.

Závislosť ide z vonkajších vrstiev do vnútorných to znamená že UI je závislé na doméne ale nie naopak. [1]

Dátova vrstva

WordEntity-reprezentuje tabuľku words v databáze, obsahuje word_id, word, translated,

Category, language

	word_id	word	translated	category	language
	Filter	Filter	Filter	Filter	Filter
1	1	CestovateI	Traveler	Cestovanie	English
2	2	Cestovná kancelária	Travel agency	Cestovanie	English
3	3	Cestovný pas	Passport	Cestovanie	English
4	4	Colnica	Customs	Cestovanie	English

Obrázok 2. Štruktúra databázy

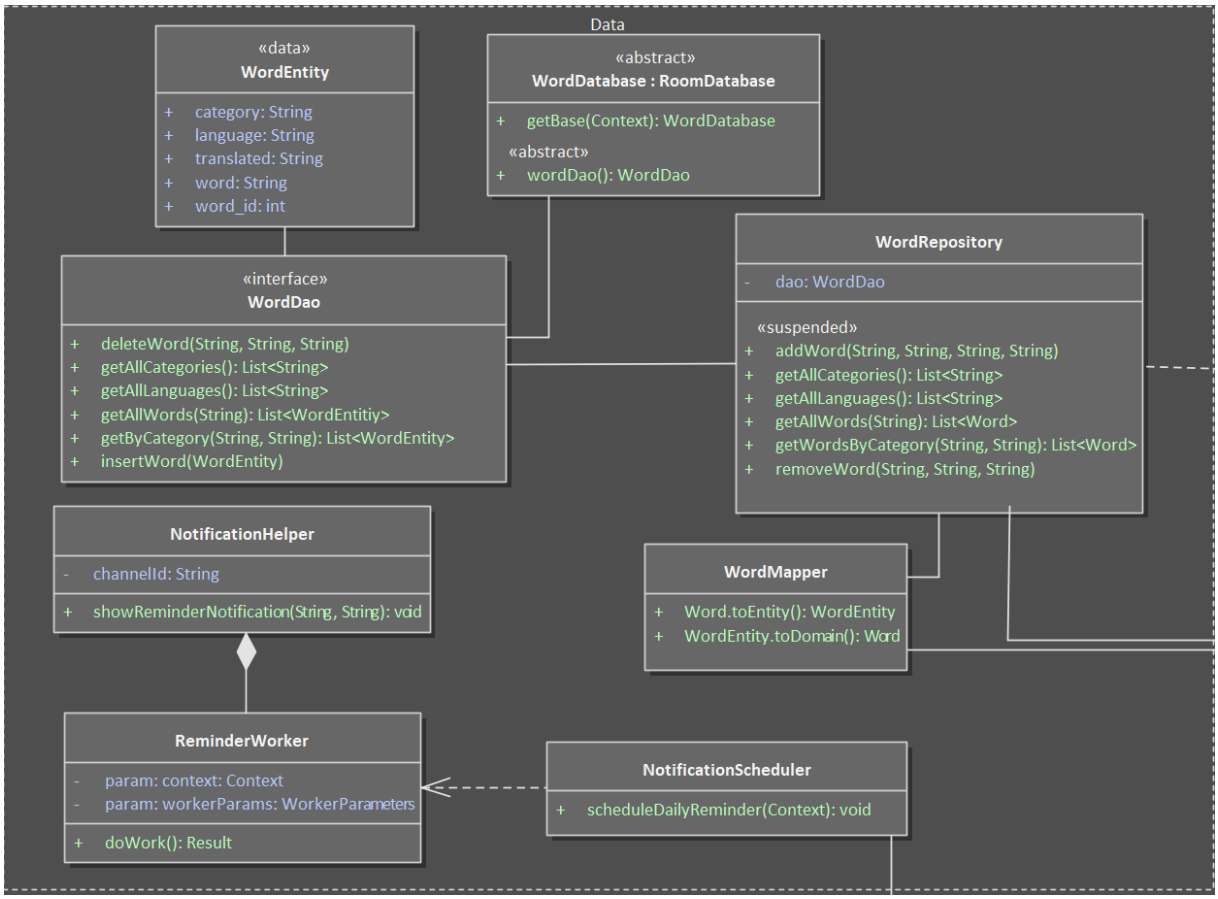
WordDao - definuje sql prístupové metódy k databáze ak sú napríklad: getAllWords(language), insertWord(wordEntity), deleteWord(word, category, language) toto robí za pomoci anotácie z Room knižnice (@Query, @Insert, @Dao...)

WordDatabase - vytvára a poskytuje inštanciu databázy, registurje WordDao

WordRepository - Implementuje doménové rozohranie WordFuncitons, používa WordDao na získavanie a ukladanie dát z a do databázy. Využíva WordMapper na mapovanie medzi WordEntity a domenovým modelom Word.

WordMapper - ma funkcie ako toDomain(), ktorá prevádza z databázovej entity na doménový model a toEntity() prevádza z doménového modelu na databázovú entitu.

Ako môžeme vidieť na obrázku, tu sa nachádzajú triedy, ktoré pracujú s databázou a triedy, ktoré spracovávajú notifikácie.



Obrázok 3. Dátová Vrstva

V dátovej vrstve sa nachádza aj práca s notifikáciami, a to konkrétne triedy NotificationHelper , ReminderWorke a NotificationScheduler

NotificationHelper – slúži ako pomocná trieda na vytváranie notifikácii a zobrazovanie notifikácii, nastaví text a ikonu notifikácii. Táto trieda požaduje oprávnenie POST_NOTIFICATIONS

ReminderWorker - pravidelne sa spúšťa na pozadí, dedí CoroutineWorker a impementuje metódu doWork(), ktorý vytvorí inštanciu NotificationHelper, zavolá metódu showReminderNotification() a samostatné zobratenie správy uživatelovi

NotificationScheduler - v našom prípade každých 8 hodín spúšťa notifikáciu, používa WorkManager + PeriodicWorkRequest

Doménová vrstva

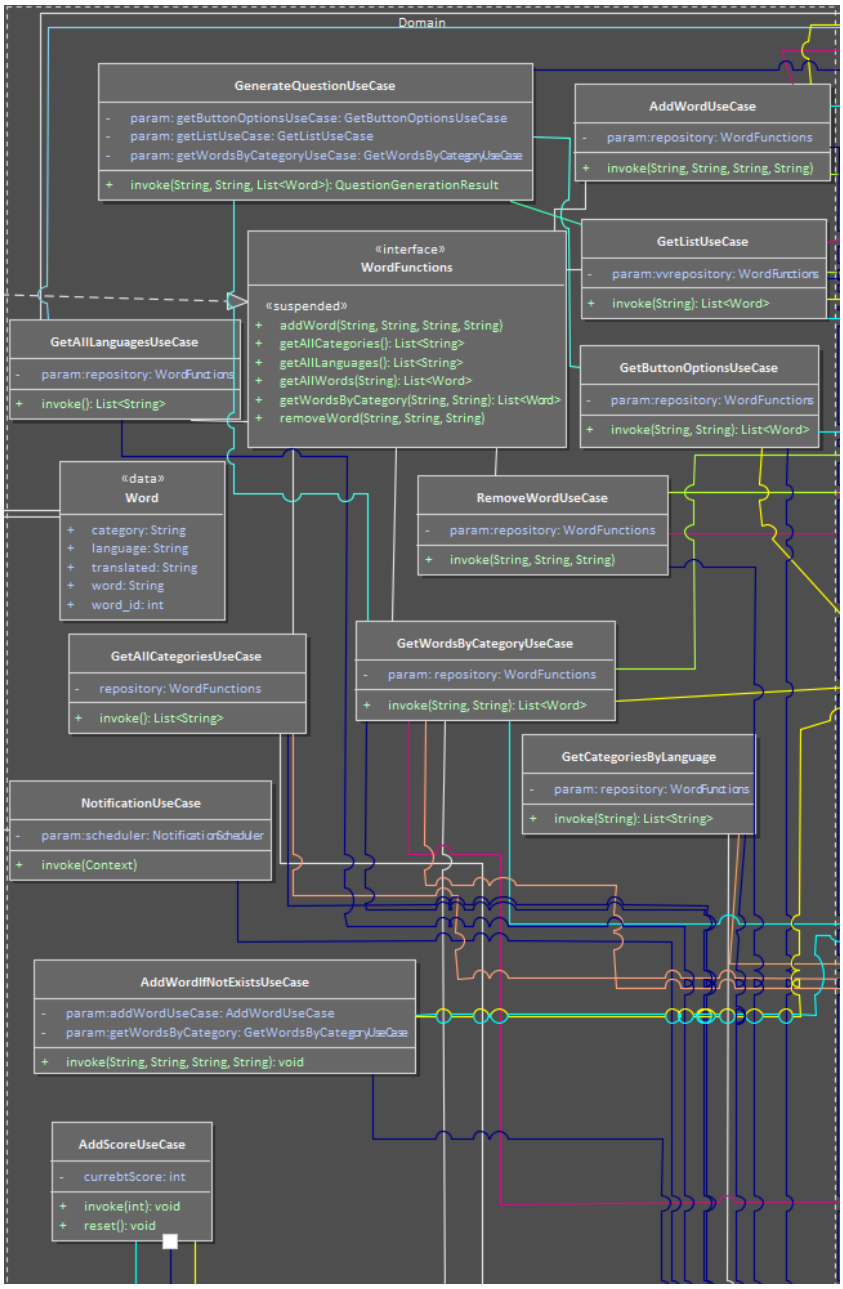
Word - dátová trieda reprezentujúca jedno slovíčko v aplikácii, obsahuje word_id, word_translated, category, language

WordFunctions - definuje všetky operácie, ktoré sú k dispozícii nad slovíčkami napríklad:

getAllWords, addWord, removeWord, getAllCategories atď. UseCas-y nikdy nekomunikujú priamo s databázou ale komunikujú cez toto rozohranie

UseCase-y - tu je doménová logika rozdelená medzi jednotlivé UseCase-y (všetky sa vytvárajú vo VocabryAplication):

1. AddScoreUseCase - stará sa o správu skóre ako je pridávanie a resetovanie skóre(používa sa v QuestionScreenViewModeli(aj Factory))
2. AddWordIfNotExistsUseCase(využíva GetWordsByCategoryUseCase a AddWordUseCase) – UseCase, ktorý pridáva slovo iba v tom prípade že už neexistuje v danej kategórii(používa sa v QuestionScreenViewModeli(aj Factory))
3. AddWordUseCase - pridáva slovo do databázy, slovo musím mať word(slovenské slovo), translation(preklad), category(kategóriu, do ktorej patrí), language(jazyk, do ktorého prekladáme) (používa sa vo WordAddingScreene (aj Factory), v usecase AddWordIfNotExistsUseSace)
4. GenerateQuestionUseCase(využíva GetListUseCase, GetWordsByCategoryUseCase, GetButtonOptionsUseCase) - usacase, ktorý kontroluje či je možné otázku vygenerovať, ak áno, tak ju vygeneruje(používa sa v QuestionScreenViewModeli (aj Factory)).
5. GetAllCategories – UseCase, ktorý vracia všetky unikátne kategórie(využíva sa v CategoryViewModel (aj Factory))
6. GetAllLanguagesUseCase – UseCase, ktorý vracia všetky unikátne jazyky(využíva sa v LanguageScreenViewModeli (aj Factory))
7. GetButtonOptions – UseCase zodpovedný za vygenerovanie možností do tlačítok(využíva sa v GenerateQuestionUseCase)
8. GetCategoriesByLanguageUseCase – UseCase, ktorý vracia všetky kategórie v danom jazyku(využíva sa v categoryViewModel (aj Factory))
9. GetListUseCase – usecase zodpovedná za získanie všetkých slov v určitom jazyku(využíva sa v GenerateQuestionUseCase, QuestionScreenViewModel (aj Factory))
10. GetWordsByCategoryUseCase – UseCase zodpovedný za vrátenie všetkých slov v danej kategórii (využíva sa v CategoryViewModeli, QuestionScreenViewModeli, WordAddingScreenViewModeli, AddWordIfNotExistsUseCase, GenerateQuestionUseCase (v každom Factory viewModeli z vymenovaných))
11. NotificationUseCase – UseCase zodpovedný za naplánovanie dennej notifikácie(využíva sa vo VocabryApplication)
12. RemoveWordUseCase – UseCase zodpovedný za odstraňovanie slov z databázy (používa sa vo WordAddingScreene)



Obrázok 4. Doménová vrstva

UI vrstva

V tejto vrstve sa nachádzajú ViewModeli, ktoré spracovávajú dáta pre obrazovky na čo ich obrazovky následne zobrazujú.

CategoryViewModel – slúži na načítanie všetkých dostupných kategórii, načítanie kategórii podľa jazyka alebo nastavenie danej kategórie za zvolenú kategóriu(používa sa vo CategorySelectScreen, WordAddingScreen, QuestionScreen, Navigation, MainActivity a Factory)

CategoryViewModelFactory – trieda, ktorá vie vytvoriť CategoryViewModel so všetkými potrebnými závislosťami (UseCase-ami) (používa sa v MainActivity)

LanguageViewModel – slúži na nastavenie jazyka ako zvoleného a načítanie všetkých jazykov(používa sa v LanguageScreene, QuestionScreene, WordAddingScreen, CategorySelectSceen, Navigation, MainActivity a Factory)

LanguageViewModelFactory – trieda, ktorá vie vytvoriť LanguageViewModel so všetkými potrebnými závislosťami (use casami) (používa sa v MainActivity)

QuestionScreenViewModel – viewModel zodpovedá za správu herného stavu, generovanie otázok, spracovanie odpovedí a pridávanie nových slov do databázy(uživatelské chyby)(používa sa v QuestionScreene, Navigáci, MainActivity a Factory)

QuestionScreenViewModelFactory – trieda, ktorá vie vytvoriť QuestionScreenViewModel so všetkými potrebnými závislosťami (Use Case-ami) (používa sa v MainActivity)

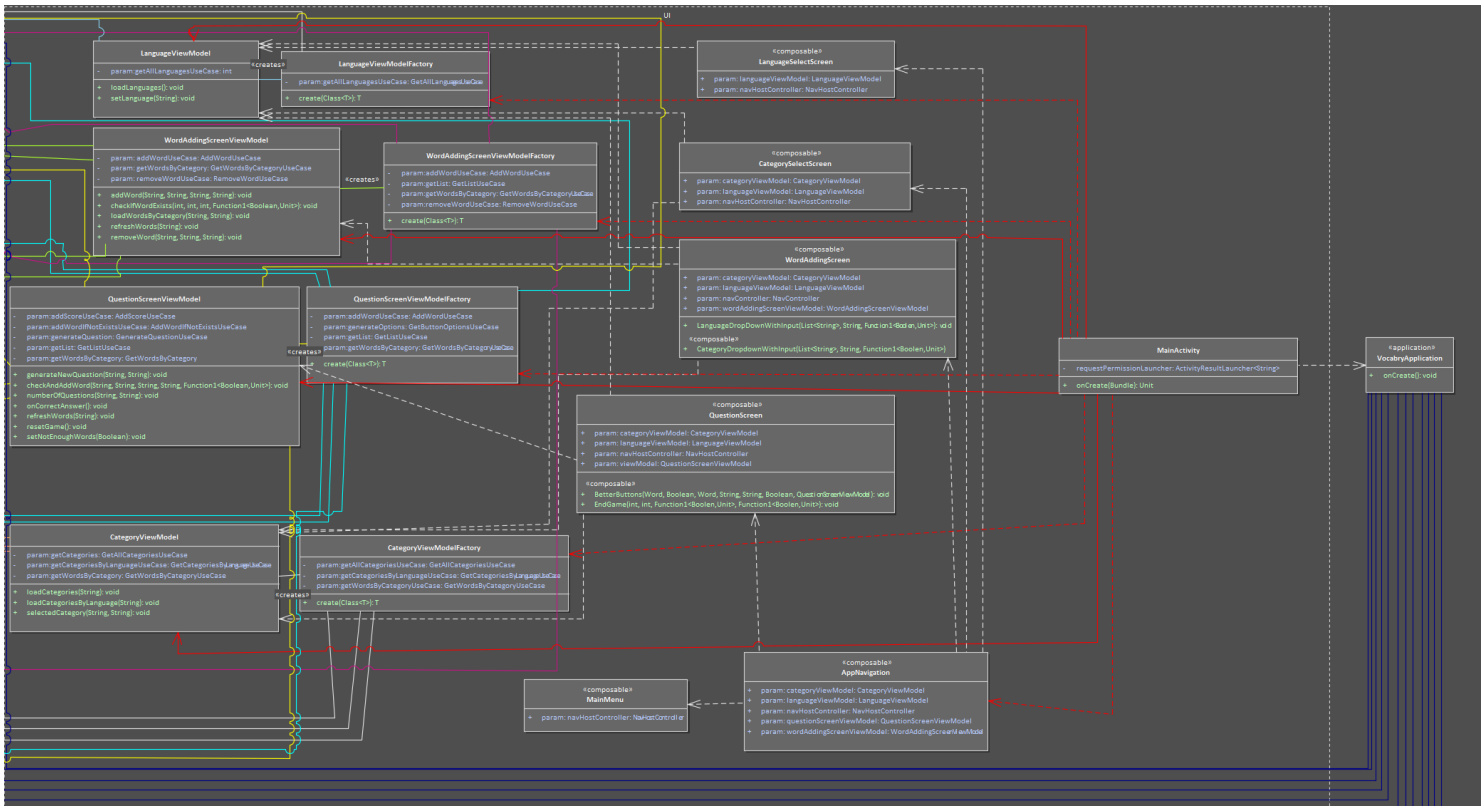
WordAddingScreenViewModel – viewModel zodpovedá za pridávanie, odoberanie, načítanie slov v danej kategórii a jazyku, kontrolovanie či slová už existujú v danom jazyku a kategórii a načítanie slov z daného jazyka a kategórie(používa sa vo WordAddingScreen, Navigácii, MainActivity a Factory)

WordAddingScreenViewModelFactory – trieda, ktorá vie vytvoriť QuestionScreenViewModel so všetkými potrebnými závislosťami (use casami) (používa sa v MainActivity)

Screens – jednotlivé obrazovky už sa starajú len o zobrazovanie dát užívateľovi a jednoduché interakcie, obrazovky pozorujú stav svojich viewModelov a reaguju na zmeny Navigation

Navigation – Navigačná logika aplikácie spravená pomocou JetPack Compose Navigation, definuje sa v nej prepínanie obrazoviek podľa routu

MainActiviy – Hlavná aktivita aplikácie, ktorá je zodpovedná za inicializáciu UI, nastavenie navigácie a odovzdanie ViewModelov do composable funkcii.



Obrázok 5. UI Vrstva

VocabryApplication – je vstupný bod aplikácie pre systém Android, slúži na inicializáciu knižníc, databáz alebo iných lokálnych konfigurácii

Obrazovky

V aplikácii sa nachádza päť hlavných obrazoviek: **MainMenuScreen**, **WordAddingScreen**, **LanguageSelectScreen**, **CategorySelectScreen** a **QuestionScreen**, pričom každá z nich má svoju vlastnú funkciu, ktorá je dôležitá pre správne fungovanie aplikácie.

MainMenuScreen

Hlavné menu, ktoré slúži ako hlavná obrazovka pre aplikáciu, je navrhnutá pomocou Jetpack Compose a zobrazuje rozohranie s prispôsobením pre orientáciu zariadenia.

Pozadie hlavného menu je riešené cez obrázok s modrým prekrytím pričom je dopĺňané s názvom aplikácie vo fonte SignikaNegative, ktorý sme si pridali.

Jednotlivé tlačidlá slúžia pre navigáciu na ostatné obrazovky, prípadne na vypnutie aplikácie

„Start“ – tlačidlo naviguje používateľa na obrazovku LanguageSelectScreen

„Words“ – tlačidlo naviguje používateľa na obrazovku WordAddingScreen

„Exit“ – tlačidlo vypne aplikáciu

Rozloženie využíva LocalConfiguration, ktorý zisťuje orientáciu zariadenia, podľa ktorej následne posúvam tlačidlá, tak aby bola aplikácia použiteľná aj na šírku.

Obrázok 6. MainMenuScreen

WordAddingScreen

Obrazovka slúži na pridávanie, zobrazovanie a správu vlastných slovíčok v aplikácii. Používateľ tu môže jednoducho vytvoriť nové slovíčko, zadať jeho preklad, priradiť ho ku konkrétnej kategórii a vybrať jazyk. Taktiež má možnosť odstrániť existujúce slovíčka.

Pridávanie slovíčka:

-slovo v pôvodnom jazyku (napr.Jablko...)

-preklad do cieľového jazyka (napr.Apple)

-kategóriu (napr. Ovocie, Zelenina...)

-jazyk (napr.Angličtina, Nemčina...)

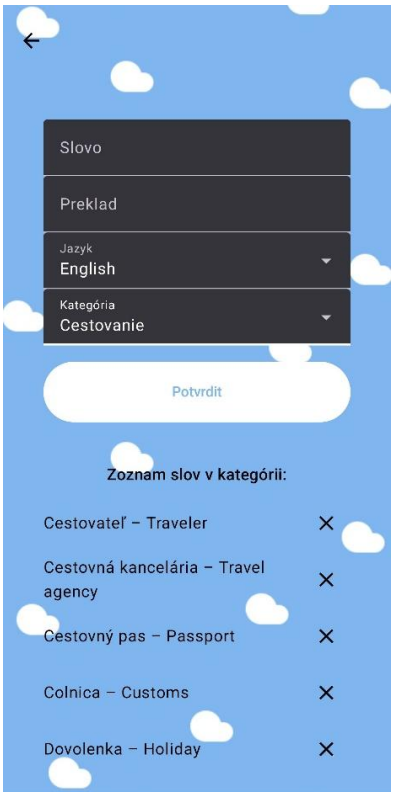
Ak zvolená kategória alebo jazyk ešte neexistujú, tak sa automaticky vytvoria.

Slovíčka sú pred pridaním kontrolované či už existujú v danej kategórii a jazyku, pričom slovo sa pridá len vtedy ak sú všetky polia vyplnené a unikátne.

Vstupné polia pre jazyk a kategóriu sú rozšírené o dropdown menu s filtrovaním podľa zadávanej hodnoty, pričom kategóriu a jazyk je možné zadať aj ručne

UI a Layout sú rovnako ako v hlavnom menu riešené pomocou compose komponentov a ich poloha a rozmery sú upravované podľa orientácie mobilu.

Obrazovka využíva rôzne ViewModeli: WordAddingScreenViewModel(pridávanie,mazanie a načítanie slov podla kategórie a jazyka), CategoryViewModel(zoznam kategórii), LanguageViewModel(dostupné jazyky a aktuálne zvolený jazyk)



Obrázok 7. WordAddingScreen

LanguageSelectScreen

Je obrazovka, ktorá slúži na zvolenie jazyka v ktorom sa chceme vzdelávať. Zvolený jazyk je následne uložený do LanguageViewModel a používateľ je presmerovaný na CategorySelectScreen.

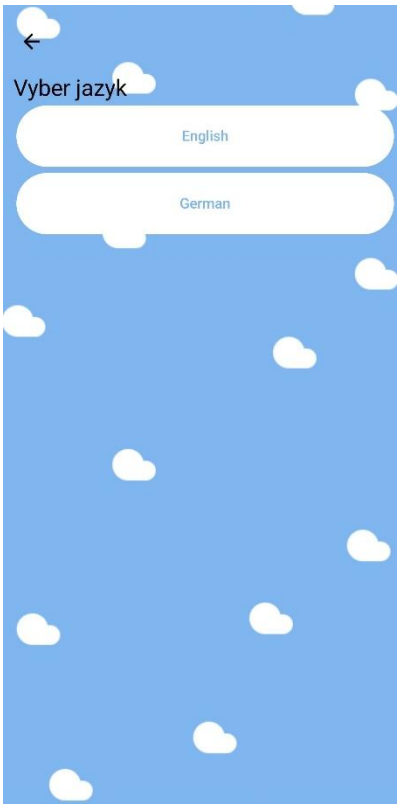
Pri načítaní obrazovky sa cez LaunchedEffect automaticky načítajú jazyky z LanguageViewModel, pričom používateľovi sa jazyky zobrazia vo forme tlačidiel. Po kliknutí na tlačidlo sa daný jazyk nastaví ako zvolený (uloží sa do ViewModelu) a používateľ je presmerovaný na ďalšiu obrazovku

Využíva LanguageViewModel, ktorý:

-zoznam dostupných jazykov (languages)

-aktuálne zvolený jazyk(selectedLanguage)

-funkcie na načítanie a nastavenie jazyka(loadLanguages, setLanguage)



Obrázok 8. LanguageSelectScreen

CategorySelectScreen

Obrazovka umožňuje používateľovi vybrať kategóriu z ktorej sa chce učiť slovíčka, a to na základe predtým zvoleného jazyka. Po výbere je používateľ presmerovaný na obrazovku so začiatkom testu

Pri načítaní obrazovky sa pomocou LaunchedEffectu automaticky načítajú všetky kategórie z CategoryViewModelu, ktoré patria k aktuálne vybranému jazyku. Po kliknutí na tlačidlo sa daná kategória nastavý ako zvolená pomocou selectedCategory a používatel je presmerovaný na obrazovku s otázkami.

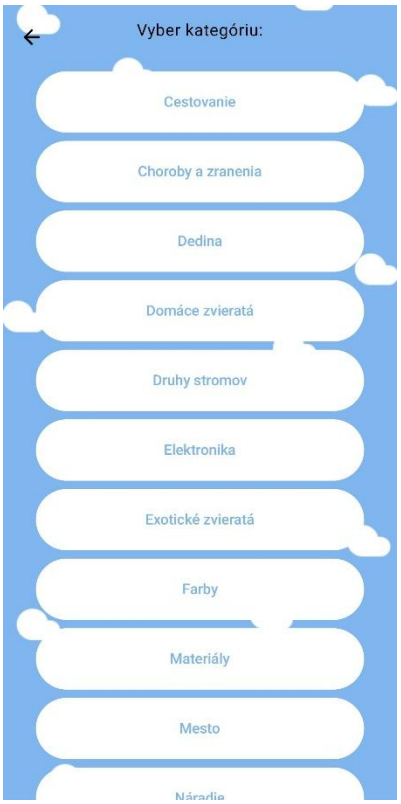
Využíva CategoryViewModel, ktorý:

-načítava kategórie podľa zvoleného jazyka(loadCategoriesByLanguage)

-uchováva vybranú kategóriu(selectedCategory)

Používa aj LanguageViewModel:

-uchováva aktuálne zvolený jazyk, podľa ktorého sa vyfiltrujú kategórie



Obrázok 9. CategorySelectScreen

QuestionScreen

Obrazovka, ktorá slúži na vzdelávanie vo forme precvičovania slovnej zásoby formou výberu správnych prekladov slov. Zobrazuje náhodne generované otázky na základe zvolenej kategórie a jazyka, pričom každá obsahuje správne slovo, ktoré je aj v otázke aj v jednom z tlačítok a 3 náhodne vybraté slová z danej kategórie.

Pri načítaní sa pomocou LaunchedEffectu načíta otázku a spočíta počet otázok v danej kategórii. Otázky sú generované na základe dát v databáze, pričom sú filtrované podľa kategórii a jazyka.

Pri nedostatku slov na výber do tlačidiel z danej kategórie sa vyberajú slová z ostatných kategórii no je ošetrená aj možnosť kedy sa v celom jazyku nachádza menej slov ako 4 čo spôsoby to že bude užívateľ presmerovaný do WordAddingScreenu aby pridal ďalšie slovíčka.

Pri odpovedi sú odpovede rozdelené na správne a nesprávne, kde správne sú signalizované zeleným rozsvietením tlačidla a pridaním skóre a zlá odpoveď je signalizovaná červeným rozsvietením tlačidla, kde sa hádané slovíčko následne pridá do kategórie „Chyby“. Tlačidlá s meniacim farbami sú urobené ako vlastný komponent BetterButtons().

Po dokončení kategórie (prejdení všetkých slovíčok) na užívateľa vyskočí text s jeho finálnym výsledkom a 2 tlačidlá, jedno ktoré reštartuje danú kategóriu a pôjdete odznovu alebo druhé, ktoré nás vráti do MainMenuScreenu. Toto všetko je zabezpečené vlastným komponentom EndGame()

Využíva QuestionScreenViewModel, ktorý:

-obsahuje herné dáta ako otázky, skóre a správne odpovede

-obsahuje logiku pre addScore, genereteNewQuestion, checkIfWordExists

Využíva aj CategoryViewModel a LanguageViewModel:

-uchovávajú aktuálne zvolený jazyk a kategóriu

Obrázok 10. QuestionScreen

AndroidX komponenty

LifeCycle

LifeCycle je pojem, ktorý označuje životný cyklus komponentov ako sú Activity alebo ViewModel. V skratke to znamená že každý komponent prechádza rôznymi stavmi a Android ich automaticky riadi. [2]

V našom prípade používame lifeCycle vo všetkých ViewModeloch (CategorySelectViewModel, LanguageSelectViewModel, QuestionScreenViewModel, WordAddingScreenViewModel) a to za pomoci viewModelScope, ktorý zabezpečuje, že ak je nejaká korutina spustená a ViewModel sa vymaže, tak korutina sa automaticky zruší čo predchádza únikom pamäte. Ale lifeCycle sa používa aj v MainActivity a to napríklad metódou onCreate(), ktorú používame

ViewModelScope sa používa keď je funkcia suspend pričom mi používame iba suspend funkcie lebo pracujeme s dátami z databázy.

Príklady použitia v našom kóde napr. CategoryViewModel funkcia loadCategories():

```
fun loadCategories() {
    viewModelScope.launch {
        _categories.value = getCategories()
    }
}
```

Navigation

Navigation slúži na deklarovanie riadenia navigácie medzi obrazovkami . V našej práci využívame navigáciu pomocou NavHostControllera, ktorý slúži ako riadiaci prvok navigácie.

Pomocou NavHost() definujeme aké obrazovky máme, ako sa volajú a čo sa má zobraziť, jednotlivé obrazovky sú composable().

Cesty mám deklarované na jednom mieste a to pomocou enumu AppScreen

Využívame konkrétne príkazy navController.navigate() a pre návrat navController.popBackStack().

```
fun AppNavigation(
      navController: NavHostController
      languageViewModel:  LanguageViewModel
) {
   NavHost(
      navController = navController,
      startDestination = AppScreen.Menu.route,
      modifier = modifier
    ) {
        composable(AppScreen.Menu.route){
   MainMenu(navController)
}
    composable(AppScreen.Language.route){

LanguageSelectScreen(languageViewModel,navController)
}
```

A v obrazovke už sa použije napríklad : navController.navigate(AppScreen.Menu.route)

Room

V aplikácii používane komponent Room na implementáciu lokálnej databázy, ktorá slúži na ukladanie a načítavanie slovíčok, ich prekladov, kategórii a jazykov. Room poskytuje

jednoduché rozohranie pre pácu so SQLite a zároveň umožňuje pracovať s objektami namiesto SQL príkazov.

V našej triede máme viacero tried, ktoré aplikácii dopomáhajú pri komunikácii s databázou, prvá z nich je **WordEntity,** ktorá reprezentuje tabuľku words v databáte

```
@Entity(tableName = "words")
data class WordEntity(
    @PrimaryKey(autoGenerate = true)
    val word_id: Int = 0,
    val word: String,
    val translated: String,
    val category: String,
    val language: String
)
```

Ďalšia trieda je **WordDao,** ktorá obsahuje všetky databázové operácie ako je napr vkladanie(insertWord), odstraňovanie(deleteWord) atď napríklad:

```
@Insert(onConflict = OnConflictStrategy.IGNORE)
suspend fun insertWord(word: WordEntity)
```

Trieda, ktorá zabezpečuje prístup a inicializáciu samostatnej databázy sa volá **WordDatabase.** Trieda vytvára Room databázu WordDatabase, ktorá obsahuje tabuľku definovanú v WordEntity a umožňuje prístup cez WordDao. Na čo sa následne používa singleton, aby existovala len jedna inštancia databázy v celej aplikácii. Databáza sa načíta z predpripraveného súboru „WordsDatabase.db“ uloženom v assetoch, vďaka čomu môže aplikácia pracovať s preddefinovanými dátami hneď po nainštalovaní.

A posledná trieda, ktorá sa zaobera roomom je trieda **WordRepository**, ktorá implementuje rozohranie WordFunctions, komunikuje s DAOm a používa sa v UseCase vrstvách pre prístup k dátam.

ViewModel

V našej aplikácii využívame komponent ViewModel, ktorý nám pomáha oddeliť zodpovednosť medzi logikou a používateľským rozohraním. Každá obrazovka, ktorá vyžaduje prácu s dátami má vlastný ViewModel, v ktorom sú spravované operácie, premenné a komunikácia s dátovou vrstvou cez UseCase. [3]

ViewModel nám pomáha pri uchovávaní stavov napríklad pri otočení obrazovky, čo zabezpečuje že stav sa nemusí znova načítať ale ostane zachovaný. [3]

Naše ViewModely:

QuestionScreenViewModel (generovanie otázok, spravovanie skóre, overovanie duplicít,

WordAddingScreenViewModel (pridávanie, odoberanie slov, overovanie duplicít, načítanie slov podľa kategórie)

CategoryViewModel (uchováva zoznam kategórii a umožňuje ich filtrovanie podľa jazyka, uchováva a aktualizuje aktuálne zvolenú kategóriu)

LanguageViewModel (uchováva a aktualizuje aktuálne zvolený jazyk, načítava dostupné jazyky)

CategoryViewModel a **LanguageViewModel** sú použité vo viacerých screenoch lebo sú považované ako doplnkové viewModely, ktoré len uchovávajú jazyky a kategórie.

Využívame aj niečo čo sa volá ViewModel Factory čo je vlastná trieda, ktorá slúži na vytváranie ViewModelov s parametrami, ktoré nemôžu byť priamo odovzdané pomocou defaultného systému.

Funguje to, tak že factory trieda prijíma nejaké závislosti, ktoré sú neskôr odovzdané klasickému viewModelu pri jeho vytváraní. Metóda v triede factory sa volá create a vykonáva sa automaticky keď android potrebuje inštanciu viewModelu, je tam príkaz modelClass.isAssignableFrom(), ktorý kontroluje či chce človek vytvoriť práve požadovaný viewModel (napr CagetoryScreenViewModel factory bude kontrolovať či chcem vytvoriť CategoryScreenViewModel) ak áno vytvorí sa s požadovanými UseCase objekatami ak nie vyhodí sa vynímka IllegalArgumentException

```
override fun <T : ViewModel> create(modelClass: Class<T>): T {
    if (modelClass.isAssignableFrom(CategoryViewModel::class.java)) {
        return CategoryViewModel(
            getAllCategoriesUseCase,
            getWordsByCategoryUseCase,
            getCategoriesByLanguageUseCase
        ) as T
    }
    throw IllegalArgumentException("Unknown ViewModel class: ${modelClass.name}")
}
```

WorkManager

V aplikácii využívame aj WorkManager, ktorý slúži na plánovanie a spúšťanie opakovaných notifikácii na pozadí. WorkManager zabezpečuje aby sa úlohy vykonali aj po vypnutí alebo reštartovaní aplikácie.

V našej práci je zakomponovaný v triede NotificationScheduler, kde plánuje dennú pripomienku, ktorá sa spustí každých 8 hodín pomocou PeriodicWorkRequest.

```
WorkManager.getInstance(context).enqueueUniquePeriodicWork(
    context.getString(R.string.daily_reminder),
    ExistingPeriodicWorkPolicy.UPDATE,
    workRequest
)
```

Notifikácie

Notifikácie v našej aplikácii sú riešene pomocou 3 tried, ktoré navzájom spolupracujú a dopĺňajú sa

NotificationHelper – slúži ako pomocná trieda na vytváranie notifikácii a zobrazovanie notifikácii, nastaví text a ikonu notifikácii. Táto trieda požaduje oprávnenie POST_NOTIFICATIONS

ReminderWorker - pravidelne sa spúšťa na pozadí, dedí CoroutineWorker a impementuje metódu doWork(), ktorý vytvorí inštanciu NotificationHelper, zavolá metódu showReminderNotification() a samostatné zobratenie správy uživatelovi

NotificationScheduler - v našom prípade každých 8 hodín spúšťa notifikáciu, používa WorkManager + PeriodicWorkRequest

Bibliografia

[1] Google, „Guide to app architecture,“ 10 Február 2025. [Online]. Available: https://developer.android.com/topic/architecture?continue=https%3A%2F%2Fdeveloper.android.com%2Fcourses%2Fpathways%2Fandroid-architecture%23article-https%3A%2F%2Fdeveloper.android.com%2Ftopic%2Farchitecture.

[2] Google, „Use Kotlin coroutines with lifecycle-aware components,“ 10 Február 2025. [Online]. Available: https://developer.android.com/topic/libraries/architecture/coroutines.

[3] Google, „ViewModel overview,“ 10 Február 2025. [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel.