ProjektLife – habit tracker

Popis a analýza riešeného problému

V živote sa ľudia často stratia a nevedia presne čo a kedy robili alebo ako to môže ovplyvniť ich život, často máme veľa zlozvykov problémov a nedostatkov o ktorých nevieme. Neexistuje dobré miesto pre jednoduchú správu a prehľad našich, denných zvykov. Aplikácie ktoré tento problém riešia bežne majú neprívetivú cenu.

Aplikácie ktoré riešia podobný problém:

Streaks

Streaks je oceňovaná aplikácia pre zoznam úloh, ktorá vám pomáha aj budovať návyky. Môžete sledovať až 24 návykov naraz, s cieľom nikdy nevynechať deň a udržať svoj "streak". Tento nástroj, dostupný len pre produkty Apple, sa spája s údajmi o zdraví vášho zariadenia, čo z neho robí skvelú voľbu pre tých, ktorí sledujú fyzickú aktivitu alebo iné fitness ciele.

Cena: 4,99 USD mesačne

HabitNow

HabitNow má užívateľsky prívetivú, komplexnú aplikáciu, ktorá vám pomáha nastaviť si vlastné ciele, spravovať úlohy a vizualizovať pokrok. Môžete organizovať svoj život a sledovať návyky bez nutnosti prechádzať medzi kalendárom a sledovačom návykov. Aplikácia tiež ponúka pripomienky a alarmy, ktoré vás povzbudia k sledovaniu procesu formovania návykov.

Cena: Zadarmo pre 7 návykov; 9,99 USD za Premium (jednorazový poplatok)

V čom sa líši projektLife

Jednoduché kategorizovanie, pridávanie návykov ako aj aktivít v neobmedzenom množstve, možnosť jednorazovej aktivity/cieľa, štatistické zobrazenie aktivít, jednoduchá správa aktivít a kategórií

Návrh riešenia problému:

Tento problém sa dá riešiť tak že vytvoríme jednoducho ovládateľnú aplikáciu v ktorej sa budú zaznamenávať denné zvyky a ciele, jednoduchým spôsobom. Toto sa dá docieliť kategorizovaním aktivít a ich ďalším zaznamenávaním. Neskoršie napravovanie a zobrazovanie zvykov vytvoríme takým spôsobom, že zobrazíme užívateľovi graf ktorý bude ukazovať denné štatistiky a graf ktorý zobrazí progres užívateľa postupom času. Pre jednoduchosť pridáme úpravu kategórií a ich vymazávanie. Takisto pridáme aj vymazávanie aktivít.

Opis súborov v projekte:

Databázové súbory

AktivitaDao.kt

Definuje DAO rozhranie pre entitu Aktivita, poskytujúce základné CRUD operácie (vytvorenie, čítanie, aktualizácia, mazanie) a ďalšie špecifické dotazy pre túto entitu.

Databaza.kt

Definuje hlavný databázový súbor, ktorý inicializuje databázu Room a poskytuje prístup k DAO rozhraniam pre jednotlivé entity (AktivitaDao, KategoriaDao, UlozeneDao).

KategoriaDao.kt

Definuje DAO rozhranie pre entitu Kategoria, poskytujúce základné CRUD operácie a špecifické dotazy pre túto entitu.

UlozeneDao.kt

Definuje DAO rozhranie pre entitu Ulozene, poskytujúce základné CRUD operácie a dotazy podľa dátumového rozsahu.

Dátové triedy

Aktivita.kt

Definuje dátovú triedu Aktivita ako entitu v databáze Room, vrátane atribútov a cudzieho kľúča na entitu Kategoria.

Kategoria.kt

Definuje dátovú triedu Kategoria ako entitu v databáze Room, vrátane základných atribútov ako nazov, farba a typ.

Ulozene.kt

Definuje dátovú triedu Ulozene ako entitu v databáze Room, vrátane atribútov a cudzích kľúčov na entity Aktivita a Kategoria.

Enum trieda

Typ.kt

Definuje enum Typ pre rôzne typy kategórií (POSITIVNA, NEGATIVNA, NEUTRALNA).

UI súbory

AktivitaScreen.kt

Definuje obrazovku pre pridanie novej aktivity, vrátane formulára s poľami pre názov, váhu, výber kategórie a typ aktivity.

KategoriaScreen.kt

Definuje obrazovku pre zobrazenie zoznamu kategórií s možnosťou pridania, úpravy a mazania kategórií.

KategoriaEditScreen.kt

Definuje obrazovku pre úpravu existujúcej kategórie, vrátane formulára na aktualizáciu názvu, farby a typu kategórie.

KategorieUpravaScreen.kt

Definuje obrazovku pre správu kategórií, vrátane zoznamu kategórií s možnosťou úpravy a mazania, a dialógového okna pre potvrdenie vymazania.

MainScreen.kt

Hlavná obrazovka aplikácie (obsah súboru nebol poskytnutý).

NastaveniaScreen.kt

Definuje obrazovku pre nastavenia aplikácie, vrátane možností vymazania všetkých dát, kategórií, aktivít a histórie.

StatisticsScreen.kt

Definuje obrazovku pre zobrazenie štatistík, vrátane grafu a zoznamu agregovaných dát na základe vybraného dátumu.

Repozitáre

KategorieRepository.kt

Implementuje repozitár pre správu kategórií, poskytuje metódy na získanie všetkých kategórií, vloženie, aktualizáciu a mazanie kategórií.

UlozeneRepository.kt

Implementuje repozitár pre správu uložených položiek, poskytuje metódy na získanie všetkých uložených položiek, vloženie a mazanie položiek.

AktivitaRepository.kt

Implementuje repozitár pre správu aktivít, poskytuje metódy na získanie všetkých aktivít, vloženie, aktualizáciu a mazanie aktivít, a metódu na získanie kategórie podľa ID.

ViewModel triedy

KategoriaView.kt

Implementuje ViewModel pre správu kategórií, poskytuje metódy na pridanie, aktualizáciu, mazanie kategórií a získanie všetkých kategórií.

UlozeneView.kt

Implementuje ViewModel pre správu uložených položiek, poskytuje metódy na pridanie, získanie všetkých uložených položiek, mazanie histórie a získanie položiek podľa dátumového rozsahu.

AktivitaView.kt

Implementuje ViewModel pre správu aktivít, poskytuje metódy na pridanie, aktualizáciu, mazanie aktivít a získanie všetkých aktivít spolu s ich kategóriami.

Hlavná aktivita

MainActivity.kt

Definuje hlavnú aktivitu aplikácie, ktorá inicializuje databázu a nastavuje navigáciu medzi jednotlivými obrazovkami pomocou Jetpack Compose.

Factory

DatabaseFactory.kt

Implementuje ViewModelProvider.Factory na vytváranie inštancií ViewModel s príslušnými repozitármi.

Hlbší opis kódu:

Hlavné body:
Aplikácia správne reaguje na otáčanie displeja použitím vytvorenej funkcie isLandscape, ktorá je typu boolean a hovorí o tom či je mobil otočený horizontálne alebo vertikálne

```
@Composable
fun isLandscape(): Boolean {
    val configuration = LocalConfiguration.current
    return configuration.orientation == Configuration.ORIENTATION_LANDSCAPE
}
```

Funkcia berie lokálnu konfiguráciu, a opýta sa jej či je telefón otočený na landscape a podľa toho vráti pravda lož.

Túto funkciu využívajú obrazovky aby si posunuli správne prvky nech sú všetky viditeľné a ovládateľné.

```
if (isLandscape()) {
    Row(
else {
    LazyColumn(
```

Funkciu týmto spôsobom využívajú obrazovky, napríklad pri zmenení orientácie zmení Row na LazyColumn čím zlepší ovládateľnosť a zobrazovanie prvku.

```
var nazov by rememberSaveable { mutableStateOf("") }
```

Ďalej prvky využívajú stav rememberSaveable čím sa zachováva hodnota prvkov aj pri zmenení stavu aplikácie čiže aj pri zmene orientácie obrazovky.

Pri špecifických častiach kde by nefungoval rememberSaveable bolo potrebné použiť takzvaný Saver ktorý funguje tak že rozloží kategóriu na mapu dvoch Stringov, používa dve funkcie save a restore save rozloží kategóriu na jej detailné prvky a tú potom použije v remember saveable,

potom ju rozloží naspäť na kategóriu, čím umožní používanie kategórie ako dataclass a zároveň jej ukladanie v remember saveable.

Aplikácia používa 7 obrazoviek: AktivitaScreen.kt, KategoriaEditScreen.kt, KategoriaScreen.kt ,KategorieUpravaScreen.kt, MainScreen.kt, NastaveniaScreen.kt, StatisticsScreen.kt

Všetky sa niečím líšia až na obrazovky KategoriaScreen a EditScreen, ktoré sú si veľmi podobné svojou funkciou. EditScreen slúži na preberanie dát vytvorených KategoriaScreen a ich úpravou.

Aplikácia používa dva externé frameworky na zobrazovanie koláčového a čiarového grafu:
com.github.tehras.charts.piechart na koláčový graf
import com.github.mikephil.charting na čiarový graf

Z androidx komponentov využíva:
Lifecycle.viewmodel – na zobrazovanie view modelov používaných room databázov

Navigation – pre navigáciu medzi obrazovkami a NavHostControlleru pre ovládanie obrazoviek z ostatných obrazoviek a presúvanie medzi obrazovkami.

Room – pre tvorenie lokálnej databázy

Viewmodel – na ovládanie funkcií Repozitárov ktoré preberajú funkcie z DAO ktoré priamo ovláda, entity ROOM databázy

Funkcia ROOM databázy
Room databáza využíva tieto nasledovný princíp ktorý vyžaduje Entity, DAO, Repository, DatabaseFactory a databázu

```
@Entity(
    tableName = "aktivita",
    foreignKeys = [ForeignKey(
        entity = Kategoria::class,
        parentColumns = ["id"],
        childColumns = ["kategoriaId"],
        onDelete = ForeignKey.CASCADE
    )]
)
data class Aktivita(
    @PrimaryKey(autoGenerate = true) val id: Int = 0,
    val nazov: String,
    val kategoriaId: Int,
    val vaha: Int,
    val jednorazova: Boolean
)
```

Takto vyzerá entita v aplikácii preberá cudzí kľúč entity Kategória, ktorý potom neskôr využíva na priradenie sa ku danej kategórii, cudzí kľúč je to preto lebo aktivita nemôže bez kategórie existovať . Hodnota autogenerate generuje kľúč pre daný prvok ten je potrebný pre unikátnosť prvku v databáze. Zvyšné časti reprezentujú len typ premennej jedného prvku a jeho názov.

Dao trieda

```
@Dao
interface KategoriaDao {
    @Query("SELECT * FROM Kategoria")
    suspend fun getAllKategorie(): List<Kategoria>

    @Query("SELECT * FROM Kategoria WHERE id = :id")
    suspend fun getKategoriaById(id: Int): Kategoria?

    @Insert
    suspend fun insertAll(vararg kategorie: Kategoria)

    @Update
    suspend fun updateKategoria(kategoria: Kategoria)

    @Delete
    suspend fun deleteKategoria(kategoria: Kategoria)

    @Query("DELETE FROM Kategoria")
    suspend fun deleteAll()
}
```

Dao trieda slúži ako interface na ovládanie tabuľky v databáze a využíva SQL príkazy na získanie prvkov. Táto trieda sa ďalej použije v repozitáry(Repository).

Repository

```
class KategorieRepository(private val kategoriaDao: KategoriaDao) {

    suspend fun getAllKategorie(): List<Kategoria> {
        return kategoriaDao.getAllKategorie()
    }

    suspend fun getKategoriaById(id: Int): Kategoria? {
        return kategoriaDao.getKategoriaById(id)
    }

    suspend fun insertAll(vararg kategorie: Kategoria) {
        kategoriaDao.insertAll(*kategorie)
    }

    suspend fun updateKategoria(kategoria: Kategoria) {
        kategoriaDao.updateKategoria(kategoria)
    }

    suspend fun deleteKategoria(kategoria: Kategoria) {
        kategoriaDao.deleteKategoria(kategoria)
    }

    suspend fun deleteAll() {
        kategoriaDao.deleteAll()
    }
}
```

Repository preberá funkcie DAO triedy a ďalej špecifikuje čo s nimi urobiť, pri vracaní dát špecifikuje aké dáta vráti a pri vymazávaní/pridávaní špecifikuje aké dáta príjme.

ViewModel

```
class UlozeneView(private val ulozeneRepository: UlozeneRepository) :
ViewModel() {
    private val _uiState = MutableStateFlow(UlozeneUiState(ulozene =
listOf()))
    val uiState: StateFlow<UlozeneUiState> = _uiState.asStateFlow()
    private val dateFormat = SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault())
    fun addUlozene(aktivitaId: Int, kategoriaId: Int, nazov: String, farba:
String, vaha: Int, date: Date) {
        viewModelScope.launch {
            val dateString = dateFormat.format(date)
            val ulozene = Ulozene(
                aktivitaId = aktivitaId,
                kategoriaId = kategoriaId,
                nazov = nazov,
                farba = farba,
                vaha = vaha,
                date = dateString
            )
            ulozeneRepository.insertAll(ulozene)
            getUlozeneByDateRange(date, date)
        }
    }
    fun getAllUlozene() {
        viewModelScope.launch {
            val ulozene = ulozeneRepository.getAllUlozene()
            _uiState.update { it.copy(ulozene = ulozene) }
        }
    }
    fun getUlozeneByDate(date: Date) {
        viewModelScope.launch {
            val DateString = dateFormat.format(date)
            val ulozeneList =
ulozeneRepository.getUlozeneByDateRange(DateString,DateString)
            _uiState.value = _uiState.value.copy(ulozene = ulozeneList)
        }
    }
}
data class UlozeneUiState(
    val ulozene: List<Ulozene>
)
```

Viewmodel je najdetajlnejšia časť Interakcie z databázou túto časť využijeme keď komunikujeme z databázov v kóde. ViewModel už ovláda priamu konverziu údajov aby ich vedela databáza spracovať a metódy ktoré uľahčia prácu z databázov alebo prípadnú repetetívnosť kódu. Zároveň špecifikuje ui state čo je prvok ktorý nám dovolí pristupovať prvkom databázy vytvorených metódami.

Databáza

```
@Database(entities = [Aktivita::class, Kategoria::class, Ulozene::class],
version = 1)
abstract class Databaza : RoomDatabase() {
    abstract fun kategoriaDao(): KategoriaDao
    abstract fun aktivitaDao(): AktivitaDao
    abstract fun ulozeneDao(): UlozeneDao

    companion object {
        @Volatile
        private var INSTANCE: Databaza? = null

        fun getDatabase(context: Context): Databaza {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    Databaza::class.java,
                    "projektlife_database"
                ).fallbackToDestructiveMigration()
                    .build()
                INSTANCE = instance
                instance
            }
        }
    }
}
```

Trieda databáza vytvára databázu do ktorej pridáva entity. Tieto entity sú konkrétne tabuľky v databáze ku ktorým pristupujeme, ktoré majú všetky hodnoty entity.

DatabaseFactory

```
class DatabaseFactory(
    private val kategorieRepository: KategorieRepository,
    private val aktivitaRepository: AktivitaRepository,
    private val ulozeneRepository: UlozeneRepository
) : ViewModelProvider.Factory {
    override fun <T : ViewModel> create(modelClass: Class<T>): T {
        return when {
            modelClass.isAssignableFrom(KategoriaView::class.java) -> {
                KategoriaView(kategorieRepository) as T
            }
            modelClass.isAssignableFrom(AktivitaView::class.java) -> {
                AktivitaView(aktivitaRepository) as T
            }
            modelClass.isAssignableFrom(UlozeneView::class.java) -> {
                UlozeneView(ulozeneRepository) as T
            }
            else -> throw IllegalArgumentException("Unknown ViewModel class")
        }
    }
}
```

Database factory vytvára dané repozitáre, cez ktoré môžeme ďalej ovládať databázu už priamo v kóde.

Zoznam použitých zdrojov:

https://github.com/tehras/charts
https://github.com/PhilJay/MPAndroidChart/tree/master?tab=readme-ov-file#documentation
https://dr-shradhasrathod.medium.com/android-room-database-with-java-beginners-dabfbbcd46a5
https://medium.com/@mohanmanu/exploring-remembersaveable-in-android-compose-b0304df3857a