МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
федеральное государственное бюджетное образовательное учреждение высшего образования  
**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Факультет информационных систем и технологий  
Кафедра «Информационные системы»  
Дисциплина «Программирование мобильных устройств»

Лабораторная работа №3

Выполнил:  
студент гр. ПИбд-32  
Алейкин А.М.  
Проверил:  
доцент кафедры  
Филиппов А.А.

Ульяновск, 2023 г

**Задание**

Разработка экранных форм приложения и настройка навигации.

Необходимо:

1. Создать необходимые для работы приложения классы-сущности (@Entity). Минимум 2 сущности. Между сущностями должны быть связи (один-к-одному, один-ко-многим или многие-ко-многим). Обязательно должна быть сущность User для последующей реализации функции аутентификации.

2. Разработать DAO-интерфейсы (@Dao) с необходимым набором методов для создания, изменения, удаления и получения сущностей.

3. Разработать класс для работы с БД с помощью Room (RoomDatabase). Получение экземпляра данного класса должно быть реализовано с помощью шаблона проектирования Singleton.

4. Добавить callback для заполнения БД тестовым набором данных. Данные можно представить в виде db-файла, json-файла или написать программный код для заполнения БД.

5. Android не позволяет запускать методы, которые обращаются к БД, в основном (mainUI) потоке. Необходимо использовать методы асинхронного выполнения запросов к БД (rxjava, coroutines).

6. Во все экранные формы (см. ЛР №2) добавить логику для получения тестовых данных из БД вместо обращения к подготовленным спискам, массивам и т. д. Необходимо учитывать асинхронную природу соответствующих методов работы с БД.

**Выполнение**

1. Были созданы 3 сущности: пользователь, история и письмо. Связи один-ко-многим между пользователем и историей, как и между пользователем и письмом.
2. @Entity(tableName = "users")  
   data class User(  
    @PrimaryKey(autoGenerate = true)  
    val id: Int? = null,  
    @ColumnInfo(name = "login")  
    val login: String,  
    @ColumnInfo(name = "password")  
    val password: String,  
   ){  
    override fun hashCode(): Int {  
    return id ?: -1  
    }  
   }

@Entity(  
 tableName = "cards",  
 foreignKeys = [  
 ForeignKey(  
 entity = User::class,  
 parentColumns = ["id"],  
 childColumns = ["user\_id"],  
 onDelete = ForeignKey.RESTRICT,  
 onUpdate = ForeignKey.RESTRICT  
 )  
 ]  
)  
data class Card(  
 @PrimaryKey(autoGenerate = true)  
 val id: Int? = null,  
 @ColumnInfo(name = "name")  
 val name: String,  
 @ColumnInfo(name = "location")  
 val location: String,  
 @ColumnInfo(name = "mileage")  
 val mileage: Int,  
 @ColumnInfo(name = "price")  
 val price: Int,  
 @ColumnInfo(name = "image")  
 val image: Bitmap,  
 @ColumnInfo(name="user\_id")  
 val userId: Int  
){  
 override fun hashCode(): Int {  
 return id ?: -1  
 }  
}

2. Были разработаны DAO-интерфейсы для каждой сущности. В них реализован необходимый набор методов. Пример DAO-интерфейса для сущности:

@Dao  
interface UserDao {  
 @Query("select \* from users")  
 fun getAll(): Flow<List<User>>  
  
 @Query("select \* from users where users.id = :id")  
 fun getById(id: Int): Flow<User?>  
  
 @Query("select \* from users where users.login = :login")  
 suspend fun getByLogin(login: String): User?  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insert(user: User)  
  
 @Update  
 suspend fun update(user: User)  
  
 @Delete  
 suspend fun delete(user: User)  
  
 @Query("SELECT \* FROM users WHERE login = :login AND password = :password")  
 fun getUserByLoginAndPassword(login: String, password: String): User?  
  
}

@Dao  
interface CardDao {  
 @Query("select \* from cards")  
 fun getAll(): PagingSource<Int, Card>  
  
 @Query("select \* from cards where cards.id = :id")  
 fun getById(id: Int): Flow<Card?>  
  
 @Query("select \* from cards where cards.user\_id = :userId")  
 fun getByUserId(userId: Int): PagingSource<Int, Card>  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insert(card: Card)  
  
 @Update  
 suspend fun update(card: Card)  
  
 @Delete  
 suspend fun delete(card: Card)  
}

3. Был разработан абстрактный класс для заполнения БД набором тестовых данных MobileAppDataBase. Заполнение БД через программный код. Использовались методы асинхронного выполнения запросов к БД coroutines.

@Database(entities = [User::class, Card::class], version = 1, exportSchema = false)  
@TypeConverters(ImageConverter::class)  
abstract class MobileAppDataBase : RoomDatabase() {  
 abstract fun userDao(): UserDao  
 abstract fun cardDao(): CardDao  
  
 companion object {  
 private const val DB\_NAME: String = "my-db.db"  
  
 @Volatile  
 private var INSTANCE: MobileAppDataBase? = null  
  
 fun getInstance(appContext: Context): MobileAppDataBase {  
 return INSTANCE ?: *synchronized*(this) **{** Room.databaseBuilder(  
 appContext,  
 MobileAppDataBase::class.*java*,  
 DB\_NAME  
 )  
 .addCallback(object : Callback() {  
 override fun onCreate(db: SupportSQLiteDatabase) {  
 super.onCreate(db)  
 *CoroutineScope*(Dispatchers.IO).*launch* **{** initialDataBase(appContext)  
 **}** }  
 })  
 .build()  
 .*also* **{** INSTANCE = **it }  
 }** }  
  
 private suspend fun initialDataBase(appContext: Context) {  
 INSTANCE?.*let* **{** database **->** val userDao = database.userDao()  
 userDao.insert(User(id = 1, login = "Margen", password = "1234",))  
 userDao.insert(User(id = 2, login = "Leonel Messi", password = "4321",))  
  
 val cardDao = database.cardDao()  
 cardDao.insert(  
 Card(  
 name = "Феррари Имба",  
 location = "г. Ульяновск",  
 image = BitmapFactory.decodeResource(  
 appContext.*resources*,  
 R.drawable.*ferrari\_laferrari\_car2* ),  
 mileage = 7322,  
 price = 125000,  
 userId = 1  
 )  
 )  
 cardDao.insert(  
 Card(  
 name = "Феррари Два",  
 location = "г. Ульяновск",  
 image = BitmapFactory.decodeResource(  
 appContext.*resources*,  
 R.drawable.*ferrari\_laferrari\_car2* ),  
 mileage = 1233,  
 price = 15000,  
 userId = 2  
 )  
 )  
 **}** }  
 }  
}

4. Во все экранный формы была добавлена логика для получения данных из БД. Пример логики получения данных по карточкам:

|  |
| --- |
| LaunchedEffect(Unit) { |
|  | withContext(Dispatchers.IO) { |
|  | val database = MobileAppDataBase.getInstance(context) |
|  | database.cardDao().getAll().collect { data -> |
|  | cards.clear() |
|  | data.forEach { cards.add(it) } |
|  | } |
|  | } |
|  | } |

**}  
}**

5. Демонстрация работы приложения. Внешний вид программы не изменился с прошлой лабораторной.