МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

**Лабораторная работа № 3**

*по дисциплине «Программирование мобильных устройств»*

Выполнил студен группы ПИбд-32

Преснякова В. В.

Проверил доцент кафедры

«Информационные системы»

Филиппов А.А.

Ульяновск, 2023

**Задание** Хранение данных (персистентность). Необходимо: 1. Создать необходимые для работы приложения классы-сущности (@Entity). Минимум 2 сущности. Между сущностями должны быть связи (один-к-одному, один-комногим 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. 7. Отчет и изменения проекта загрузить в репозиторий по адресу http://student.git.athene.tech

**Решение**

1. Создадим необходимые модели для работы

@Entity  
data class Hotel (  
 @PrimaryKey(autoGenerate = true)  
 val hotelId: Int? = null,  
  
 @ColumnInfo(name = "Name")  
 val name: String,  
  
 @ColumnInfo(name = "Price")//  
 val price: Double,  
  
 @ColumnInfo(name = "Img")//  
 val img: Int,  
  
 @ColumnInfo(name = "Stars")//  
 val stars: Int,  
  
 @ColumnInfo(name = "Location")//  
 val location: String,  
  
 @ColumnInfo(name = "Info")//  
 val info: String,  
)

@Entity(tableName = "users")  
data class User(  
 @PrimaryKey(autoGenerate = true)  
 val userId: Int? = null,  
 @ColumnInfo(name = "Name")  
 val name: String,  
 @ColumnInfo(name = "Surname")  
 val surname: String,  
 @ColumnInfo(name = "Email")  
 val email: String,  
 @ColumnInfo(name = "Password")  
 val password: String,  
 @ColumnInfo(name = "Role")  
 val role: RoleEnum,  
 @ColumnInfo(name = "Photo")  
 val photo: Int? = null,  
)

@Entity  
data class Order(  
 @PrimaryKey(autoGenerate = true)  
 val orderId: Int? = null,  
 @ColumnInfo(name = "DateFrom")  
 val dateFrom: String,  
 @ColumnInfo(name = "DateTo")  
 val dateTo: String,  
 @ColumnInfo(name = "Rooms")  
 val rooms: Int,  
 @ColumnInfo(name = "Total")  
 val total: Double,  
 @ColumnInfo(name = "CreatorUserId")  
 val creatorUserId: Int,  
 @ColumnInfo(name = "BookedHotelId")  
 val bookedHotelId: Int,  
 @Embedded  
 val hotel: Hotel  
)

Сделаем связи между сущностями

data class UserWithOrder(  
 @Embedded val user: User,  
 @Relation(  
 parentColumn = "userId",  
 entityColumn = "CreatorUserId"  
 )  
 val orders: List<Order>  
)

2. Создадим Dao интерфейсы для взаимодействия с сущностями

@Dao  
interface HotelDao {  
 @Insert  
 suspend fun insert(hotel: Hotel)  
  
 @Update  
 suspend fun update(hotel: Hotel)  
  
 @Delete  
 suspend fun delete(hotel: Hotel)  
  
 @Query("SELECT\*FROM Hotel")  
 fun getAllHotels(): Flow<List<Hotel>>  
  
 @Query("SELECT \* FROM Hotel WHERE hotelId = :id")  
 suspend fun getHotelById(id: Int): Hotel  
}

@Dao  
interface OrderDao {  
  
 @Insert  
 suspend fun createOrder(order: Order): Long  
  
 @Insert  
 suspend fun insertOrderHotel(orderHotel: OrderHotel)  
  
 @Query("SELECT \* FROM 'Order' WHERE orderId = :id")  
 fun getOrderWithHotels(id: Int): Hotels  
  
 @Query("SELECT \* FROM `Order`")  
 fun getAllOrder(): Flow<List<Order>>  
  
 @Delete  
 suspend fun delete(order: Order)  
}

@Dao  
interface UserDao {  
 @Insert  
 suspend fun createUser(user: User)  
  
 @Update  
 suspend fun updateUser(user: User)  
  
 @Delete  
 suspend fun deleteUser(user: User)  
  
 @Query("SELECT \* FROM users WHERE userId = :id")  
 suspend fun getUserById(id: Int): User  
  
 @Query("SELECT \* FROM users WHERE email = :email")  
 suspend fun getUserByEmail(email: String): User  
  
 @Query("SELECT \* FROM users WHERE userId =:id")  
 fun getUserOrders(id: Int) : Flow<UserWithOrder>  
}

1. Разработаем класс для работы с БД
2. @Database(entities = [Hotel::class, User::class, Order::class, OrderHotel::class], version = 5)  
   abstract class AppDatabase : RoomDatabase() {  
    abstract fun hotelDao(): HotelDao  
    abstract fun userDao(): UserDao  
    abstract fun orderDao(): OrderDao  
     
    companion object {  
    private const val DB\_NAME: String = "my-db"  
     
    @Volatile  
    private var INSTANCE: AppDatabase? = null  
     
    suspend fun populateDatabase() {  
    INSTANCE?.*let* **{** database **->** // User  
    val userDao = database.userDao()  
    val user1 = User(null, "Artem", "Emelyanov", "artem@mail.ru", "123", RoleEnum.*Admin*)  
    val user2 = User(null, "Danil", "Markov", "danil@mail.ru", "123", RoleEnum.*User*)  
    val user3 = User(null, "Viktoria", "Presnyakova", "vika@mail.ru", "123", RoleEnum.*User*)  
    userDao.createUser(user1)  
    userDao.createUser(user2)  
    userDao.createUser(user3)  
    val hotelDao = database.hotelDao()  
   val hotel1 = Hotel(null, "Hotel1", 1000.0, R.drawable.*img*, 1, "location1", "info1")  
   val hotel2 = Hotel(null, "Hotel2", 2000.0, R.drawable.*img\_2*, 2, "location2", "info2")  
   val hotel3 = Hotel(null, "Hotel3", 3000.0, R.drawable.*img\_3*, 3, "location3", "info3")  
   val hotel4 = Hotel(null, "Hotel4", 4000.0, R.drawable.*img\_4*, 4, "location4", "info4")
3. hotelDao.insert(hotel1)  
    hotelDao.insert(hotel2)  
    hotelDao.insert(hotel3)  
    hotelDao.insert(hotel4)  
    // Order  
    **}** }  
     
    fun getInstance(appContext: Context): AppDatabase {  
    return INSTANCE ?: *synchronized*(this) **{** Room.databaseBuilder(  
    appContext,  
    AppDatabase::class.*java*,  
    DB\_NAME  
    )  
    .addCallback(object : Callback() {  
    override fun onCreate(db: SupportSQLiteDatabase) {  
    super.onCreate(db)  
    *CoroutineScope*(Dispatchers.IO).*launch* **{** populateDatabase()  
    **}** }  
    })  
    .fallbackToDestructiveMigration()  
    .build()  
    .*also* **{** INSTANCE = **it }  
    }** }  
    }  
   }
4. Так же тут присутсвует метод populateDatabase заполнения тестовыми данными
5. Для вызова методов получения подготовленных списков используем функции, которые позволяют работать с асинхронными операциями

val basketList = remember { mutableStateMapOf<Hotel, Int>() }

val context = LocalContext.current

LaunchedEffect(Unit) {

withContext(Dispatchers.IO) {

AppDatabase.getInstance(context).basketDao().getBasketWithHotels(1).collect { data ->

basketList.clear()

for (item in data.services){

basketList[item] = (basketList[item] ?: 0) + 1

}

}

}

}