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  
Бондаренко М.С.  
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.

@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,  
 @ColumnInfo(name = "email")  
 val email: String,  
 @ColumnInfo(name = "photo")  
 val photo: Bitmap? = null  
){  
 override fun hashCode(): Int {  
 return id ?: -1  
 }  
}

@Entity(  
 tableName = "stories",  
 foreignKeys = [  
 ForeignKey(  
 entity = User::class,  
 parentColumns = ["id"],  
 childColumns = ["user\_id"],  
 onDelete = ForeignKey.RESTRICT,  
 onUpdate = ForeignKey.RESTRICT  
 )  
 ]  
)  
data class Story(  
 @PrimaryKey(autoGenerate = true)  
 val id: Int? = null,  
 @ColumnInfo(name = "title")  
 val title: String,  
 @ColumnInfo(name = "description")  
 val description: String,  
 @ColumnInfo(name = "cover")  
 val cover: Bitmap,  
 @ColumnInfo(name = "postdate")  
 val postdate: Long? = Date().*time*,  
 @ColumnInfo(name="user\_id")  
 val userId: Int  
){

@Entity(  
 tableName = "mails",  
 foreignKeys = [  
 ForeignKey(  
 entity = User::class,  
 parentColumns = ["id"],  
 childColumns = ["user\_id"],  
 onDelete = ForeignKey.RESTRICT,  
 onUpdate = ForeignKey.RESTRICT  
 )  
 ]  
)  
data class Mail(  
 @PrimaryKey(autoGenerate = true)  
 val id: Int? = null,  
 @ColumnInfo(name = "message")  
 val message: String,  
 @ColumnInfo(name = "postdate")  
 val postdate: Long? = Date().*time*,  
 @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")  
 suspend fun getById(id: Int): User?  
  
 @Query("select \* from users where users.login = :login")  
 suspend fun getByLogin(login: String): User?  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insert(vararg user: User)  
  
 @Update  
 suspend fun update(user: User)  
  
 @Delete  
 suspend fun delete(user: User)  
  
 @Query("delete from stories")  
 suspend fun deleteAll()  
}

@Dao  
interface StoryDao {  
 @Query("select \* from stories order by id desc")  
 fun getAll(): PagingSource<Int, Story>  
  
 @Query("select \* from stories where stories.id = :id")  
 suspend fun getById(id: Int): Story?  
  
 @Query("select \* from stories where stories.user\_id = :userId order by stories.id desc")  
 fun getByUserId(userId: Int): Flow<List<Story>>  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insert(vararg story: Story)  
  
 @Update  
 suspend fun update(story: Story)  
  
 @Delete  
 suspend fun delete(story: Story)  
  
 @Query("delete from stories")  
 suspend fun deleteAll()  
}

@Dao  
interface MailDao {  
 @Query("select \* from mails order by id desc")  
 fun getAll(): PagingSource<Int, Mail>  
  
 @Query("select \* from mails where mails.id = :id")  
 suspend fun getById(id: Int): Mail?  
  
 @Insert(onConflict = OnConflictStrategy.IGNORE)  
 suspend fun insert(mail: Mail)  
  
 @Update  
 suspend fun update(mail: Mail)  
  
 @Delete  
 suspend fun delete(mail: Mail)  
}

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, Story::class, Mail::class, RemoteKeys::class], version = 10, exportSchema = false)  
@TypeConverters(Converters::class)  
abstract class MobileAppDataBase : RoomDatabase() {  
 abstract fun userDao(): UserDao  
 abstract fun storyDao(): StoryDao  
 abstract fun mailDao(): MailDao  
 abstract fun remoteKeysDao(): RemoteKeysDao  
  
 companion object{  
 private const val DB\_NAME: String = "my-db"  
  
 @Volatile  
 private var INSTANCE: MobileAppDataBase? = null  
  
 suspend fun initialDataBase(appContext: Context){  
 INSTANCE?.*let* **{** database **->** val userDao = database.userDao()  
 userDao.insert(User(id = 1, login = "Дзюнзи Ито", password = "1234", email = "ito@gmail.com"))  
 userDao.insert(User(id = 2, login = "Стивен Кинг", password = "4321", email = "king@gmail.com"))  
  
 val storyDao = database.storyDao()  
 storyDao.insert(Story(title = "Переулок", description = "История ужасов от Дзюнзи Ито",  
 cover = BitmapFactory.decodeResource(appContext.*resources*, R.drawable.*dzun*), userId = 1))  
 storyDao.insert(Story(title = "Чужак", description = "Знаменитая книга стивена кинга",  
 cover = BitmapFactory.decodeResource(appContext.*resources*, R.drawable.*king*), userId = 2))  
  
 val mailDao = database.mailDao()  
 for (i in 0..50){  
 if (i % 2 == 0){  
 mailDao.insert(Mail(message = "Выложил новые страницы", userId = 1))  
 }  
 else{  
 mailDao.insert(Mail(message = "Меня отменили в Твиттере", userId = 2))  
 }  
 }  
 **}** }  
  
 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)  
 **}** }  
 })  
 .fallbackToDestructiveMigration()  
 .build()  
 .*also* **{** INSTANCE = **it }  
 }** }  
 }  
}

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

LaunchedEffect(Unit) **{** GlobalUser.getInstance().getUser()?.*let* **{** user **->** if (user!!.photo != null)  
 photo.value = user!!.photo!!  
 userId.value = user!!.id!!  
 login.value = user!!.login  
 password.value = user!!.password  
 email.value = user!!.email  
 **}  
}**

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