Отчёт по практике

**«Разработка приложений для мобильных устройств»**

Задание 3

Интеграция таблицы рекордов с локальной БД и загрузкой данных из сети

Выполнил:

Студент группы 8B21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю.Д. Алексейчик

# Задание

Реализуйте систему таблицы рекордов для игры.

1. Используя библиотеку Room, создайте локальную базу данных для хранения рекордов. Сущность ScoreRecord должна содержать: id, playerName, score (очки), date.
2. Реализуйте DAO (Data Access Object) с методами для вставки нового рекорда и получения отсортированного по убыванию очков списка всех рекордов.
3. Создайте LeaderboardFragment, который отображает список рекордов из базы данных с помощью RecyclerView. Для работы с данными используйте LiveData/Flow и корутины.
4. Добавьте функцию загрузки "топ-5 мировых рекордов" с mock-сервера (используйте, например, https://jsonplaceholder.typicode.com/posts как mock-данные, преобразовав их в модель RemoteRecord). Загрузка должна происходить в фоновом потоке с использованием корутин.
5. Создайте функцию-расширение (extension function) для класса String, которая форматирует дату из строки, полученной от сервера (например, из "2023-10-25" в "25.10.2023").

# ****Исходные данные:****

1. Проектирование UI: Макет fragment\_leaderboard.xml с RecyclerView для локальных рекордов и отдельной секцией/кнопкой для загрузки топ-5.
2. Программирование на Kotlin: Определение Entity, DAO, Database для Room. Репозиторий для абстракции над источниками данных. Использование корутин (viewModelScope, Dispatchers.IO). Реализация функции-расширения для String.
3. Работа с Android Components: Настройка Room, использование Retrofit или HttpURLConnection для сетевого запроса.
4. Тестирование: Написать модульные тесты для DAO (с использованием in-memory базы данных) и для репозитория (с моками источников данных)

# Ход работы

1. Используя библиотеку Room, создайте локальную базу данных для хранения рекордов. Сущность ScoreRecord должна содержать: id, playerName, score (очки), date.
2. Реализуйте DAO (Data Access Object) с методами для вставки нового рекорда и получения, отсортированного по убыванию очков списка всех рекордов.

Room – абстракция над SQLite, предоставляющая объектно-ориентированный интерфейс для работы с локальной базой данных.

Основные компоненты Room:

* Entity (Сущность) – класс, представляющий таблицу в базе данных.

ScoreRecord.kt

// Entity для таблицы рекордов локальной БД  
@Entity(tableName = "score\_records") // указываем название таблицы  
data class ScoreRecord(  
 @PrimaryKey(autoGenerate = true) // первичный ключ  
 val id: Int = 0,  
  
 val playerName: String,  
 val score: Int,  
  
 // "yyyy-MM-dd"  
 val date: String  
)

* DAO (Data Access Object) – интерфейс с методами доступа к данным

ScoreDao.kt

// операции для работы с базой данных  
@Dao  
interface ScoreDao {  
  
 // вставить новый рекорд  
 @Insert(onConflict = OnConflictStrategy.REPLACE)  
 suspend fun insert(record: ScoreRecord): Long  
  
 // получить все рекорды, отсортированные по убыванию очков  
 @Query("SELECT \* FROM score\_records ORDER BY score DESC, date DESC")  
 fun getAllSortedByScoreDesc(): Flow<List<ScoreRecord>>  
  
 // очистить таблицу  
 @Query("DELETE FROM score\_records")  
 suspend fun clearAll()  
}

* Database (База данных)

AppDatabase.kt

// объединение таблицы и операций над таблицей  
  
// объявление базы данных (указываем ScoreRecord как таблицу)  
@Database(entities = [ScoreRecord::class], version = 1, exportSchema = false)  
abstract class AppDatabase : RoomDatabase() {  
 abstract fun scoreDao(): ScoreDao // получаем доступ к функциям из файла ScoreDao.kt  
  
 companion object {  
 // Singleton - у класса только один экземпляр  
 @Volatile // переменной для экземпляра  
 private var INSTANCE: AppDatabase? = null  
  
 // функция для получения экземпляра  
 fun getInstance(context: Context): AppDatabase {  
 // проверка на существование объекта и его создание  
 return INSTANCE ?: *synchronized*(this) **{** val instance = Room.databaseBuilder( // создание базы данных  
 context.*applicationContext*,  
 AppDatabase::class.*java*, // класс базы даннных  
 "game\_database" // имя файл базы данных  
 ).build()  
 INSTANCE = instance // сохранение созданного объекта  
 instance // возвращение объекта  
 **}** }  
  
 }  
}

1. Создайте LeaderboardFragment, который отображает список рекордов из базы данных с помощью RecyclerView. Для работы с данными используйте LiveData/Flow и корутины.

LeaderboardFragment.kt

// фрагмент, экран таблицы лидеров  
class LeaderboardFragment : Fragment() {  
  
 // получаем viewModel и адаптеры  
 private lateinit var viewModel: LeaderboardViewModel  
 private lateinit var localAdapter: ScoreAdapter  
 private lateinit var remoteAdapter: RemoteAdapter  
  
 override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?): View? {  
 return inflater.inflate(R.layout.*fragment\_leaderboard*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 viewModel = ViewModelProvider(this, ViewModelProvider.AndroidViewModelFactory.getInstance(requireActivity().*application*))  
 .get(LeaderboardViewModel::class.*java*)  
  
 // нахождение элементов  
 val rvLocal: RecyclerView = view.findViewById(R.id.*rvLocalScores*)  
 val rvRemote: RecyclerView = view.findViewById(R.id.*rvRemoteTop*)  
 val btnLoad: Button = view.findViewById(R.id.*btnLoadTop5*)  
  
 // настрока адаптеров  
 localAdapter = ScoreAdapter()  
 remoteAdapter = RemoteAdapter()  
  
 // связывание адаптеров со списками  
 rvLocal.*layoutManager* = LinearLayoutManager(requireContext())  
 rvLocal.*adapter* = localAdapter  
  
 rvRemote.*layoutManager* = LinearLayoutManager(requireContext())  
 rvRemote.*adapter* = remoteAdapter  
  
 // наблюдение за локальными рекордами через StateFlow  
 *lifecycleScope*.*launch* **{** viewModel.localRecords.collectLatest **{** list **->** localAdapter.submitList(list)  
 **}  
 }** // наблюдение за топ-5, показываем список когда придёт  
 *lifecycleScope*.*launch* **{** viewModel.remoteTop.collectLatest **{** list **->** if (list.isEmpty()) {  
 rvRemote.*visibility* = View.*GONE* } else {  
 rvRemote.*visibility* = View.*VISIBLE* // показывается, если есть данные  
 remoteAdapter.submitList(list)  
 }  
 **}  
 }** // обработка кнопки  
 btnLoad.setOnClickListener **{** // загрузка данных  
 viewModel.loadTop5()  
 Toast.makeText(requireContext(), "Загрузка топ-5...", Toast.*LENGTH\_SHORT*).show() // показваем сообщение в окошке для загрузки данных  
 **}** }  
}

Фрагмент LeaderboardFragment.kt используется для отображения игровых рекордов из локальной базы данных и с сервера. StateFlow – поток данных, который автоматически уведомляет подписчиков, когда его значение меняется. StateFlow используется для обновления таблицы, при добавлении новых рекордов или при их загрузки из сети. В качестве основного макета списка используется макет fragment\_leaderboard.xml, item\_score.xml – шаблон элемента локальных рекордов, item\_remote\_score.xml – шаблон элемента рекордов из сети. Для реализации логики RecyclerView используются файлы: ScoreAdapter – для адаптера локальных рекордов, RemoteAdapter.kt – для адаптера рекордов из сети.

1. Добавьте функцию загрузки "топ-5 мировых рекордов" с mock-сервера (используйте, например, https://jsonplaceholder.typicode.com/posts как mock-данные, преобразовав их в модель RemoteRecord). Загрузка должна происходить в фоновом потоке с использованием корутин.

Mock-сервер – сервер, который имитирует поведение реального сервера для тестирования.

ApiService.kt

// определение http запроса к серверу  
interface ApiService {  
 @GET("/posts") // делает запрос по адресу /posts, чтобы получить данные с сервера  
 // функция корутина, возвращает список объектов  
 suspend fun getPosts(): List<PostDto>  
}  
  
// DTO для jsonplaceholder posts  
data class PostDto( // шаблон для данных, которые получаем с сервера  
 val userId: Int,  
 val id: Int,  
 val title: String,  
 val body: String  
)

ApiService – описание HTTP-запросов к серверу с использованием библиотеки Retrofit. Retrofit – библиотека для Android, которая упрощает взаимодействие с HTTP-сервисами.

LeaderboardRepository.kt

/ репозиторий объединяет доступ к локальным записям и загрузку remote top-5  
class LeaderboardRepository(context: Context) { // cоздание репозитория, который работает и с локальной бд и с сервером  
  
 // получение доступк к (singleton) dao через appDatabase для работы с бд  
 private val scoreDao: ScoreDao = AppDatabase.getInstance(context).scoreDao()  
  
 // Retrofit - библиотека для http запросов  
 private val api: ApiService by *lazy* **{** Retrofit.Builder()  
 .baseUrl("https://jsonplaceholder.typicode.com") // url для api  
 .addConverterFactory(GsonConverterFactory.create()) // преобразование в json объекты  
 .build() // создание экземпляра Retrofit  
 .create(ApiService::class.*java*) // создание объекта ApiService  
 **}** // получение Flow всех локальных рекордов отсортированных по убыванию очков (методы в ScoreDao)  
 fun getLocalRecords(): Flow<List<ScoreRecord>> = scoreDao.getAllSortedByScoreDesc()  
  
 // функция чтобы вставить новый локальный рекорд  
 suspend fun insertLocalRecord(record: ScoreRecord): Long {  
 return withContext(Dispatchers.IO) **{** scoreDao.insert(record) // вызов метода из scoreDao.kt  
 **}** }  
  
 // функция отчистки  
 suspend fun clearLocal() {  
 withContext(Dispatchers.IO) **{** scoreDao.clearAll() **}** }  
  
 // загрузка топ-5 мировых рекордов из сети и преобразование их в RemoteRecord  
 suspend fun fetchTop5Remote(): List<RemoteRecord> = withContext(Dispatchers.IO) **{** try {  
 val posts = api.getPosts()  
 *println*("Данные загружены: ${posts.size} постов")  
 val sdf = SimpleDateFormat("yyyy-MM-dd", Locale.getDefault())  
 val date = sdf.format(Date())  
  
 // преобразование 5 постов в фиктивные рекорды  
 posts.*take*(5).*map* **{** post **->** val pseudoScore = (post.body.length \* 3 + post.id \* 17) % 1000  
 RemoteRecord(  
 id = post.id,  
 playerName = post.title.*take*(20),  
 score = pseudoScore,  
 date = date  
 )  
 **}** } catch (e: Exception) {  
 Log.e("LeaderboardRepository", "Ошибка при загрузке: ${e.message}", e)  
 *emptyList*() // Возвращаем пустой список, чтобы не упало приложение  
 }  
 **}**}

Репозиторий абстрагирует источники данных и предоставляет единый интерфейс для работы с ним, используется для объединения доступа к разным источникам данных: локальной базе данных, серверу. Репозиторий связан с локальной базой данной и имеет к ней прямой доступ через DAO. Он загружает данные с сервера, а также передаёт данные LeaderboardViewMode, который в свою очередь передаёт данные в фрагмент с таблицей рекордов. В LeaderboardRepository.kt реализованы загрузка и преобразование рекордов из Mock-сервера.

1. Создайте функцию-расширение (extension function) для класса String, которая форматирует дату из строки, полученной от сервера (например, из "2023-10-25" в "25.10.2023").

StringExt.kt

// форматирование даты строки yyyy-MM-dd в dd.MM.yyyy  
fun String.toDisplayDate(): String {  
 // ожидание вход вида yyyy-MM-dd  
 return try {  
 val parts = this.*split*("-")  
 if (parts.size >= 3) {  
 val year = parts[0]  
 val month = parts[1]  
 val day = parts[2]  
 "$day.$month.$year"  
 } else {  
 // если формат не тот, возвращаем исходную строку  
 this  
 }  
 } catch (e: Exception) {  
 this  
 }  
}

Extension Function – функция, которая добавляет новое поведение к существующему классу без наследования или модификации исходного кода. Так как в полученных данных в mock-сервере нет даты, функция toDisplayDate будет использоваться, когда появится локальные рекорды игрока.

В MainActivity.kt и activity\_main.xml была добавлена кнопка “Таблица рекордов” и взаимодействие с ней, для открытия фрагмента с таблицей рекордов.

Тестирование:

ScoreDaoTest.kt

@RunWith(AndroidJUnit4::class)  
class ScoreDaoTest {  
  
 private lateinit var database: AppDatabase  
 private lateinit var dao: ScoreDao  
  
 @Before  
 fun setup() {  
 database = Room.inMemoryDatabaseBuilder(  
 ApplicationProvider.getApplicationContext(),  
 AppDatabase::class.*java* ).build()  
 dao = database.scoreDao()  
 }  
  
 @After  
 fun tearDown() {  
 database.close()  
 }  
  
 @Test  
 fun insertAndRetrieveRecord() = *runBlocking* **{** // Вставляем запись  
 val record = ScoreRecord(playerName = "Test", score = 100, date = "2024-01-15")  
 dao.insert(record)  
  
 // Проверяем что запись есть в базе  
 val records = dao.getAllSortedByScoreDesc().first()  
 *assert*(records.*isNotEmpty*())  
 **}** @Test  
 fun clearAllRecords() = *runBlocking* **{** // Добавляем запись и очищаем  
 val record = ScoreRecord(playerName = "Test", score = 100, date = "2024-01-15")  
 dao.insert(record)  
 dao.clearAll()  
  
 // Проверяем что база пустая  
 val records = dao.getAllSortedByScoreDesc().first()  
 *assert*(records.isEmpty())  
 **}**}

Тест проверяет базовые операции Room DAO с использованием in-memory базы данных: вставку записей, получение данных и очистку базы.

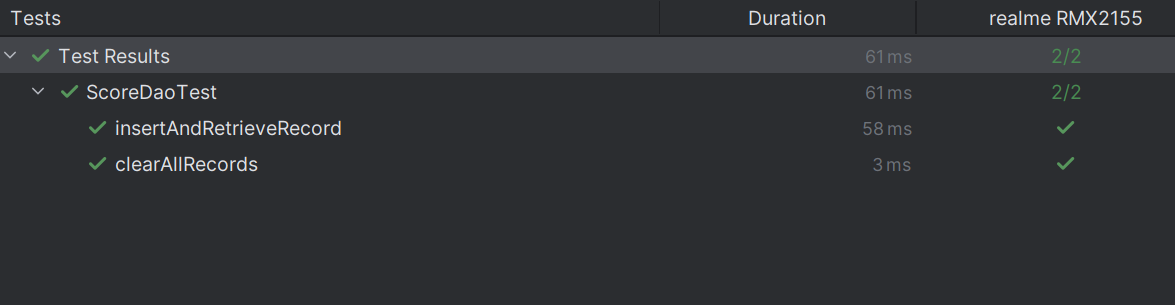


Рисунок 1. Успешно пройденный тест ScoreDaoTest

LeaderboardRepositoryTest.kt

@RunWith(AndroidJUnit4::class)  
class LeaderboardRepositoryTest {  
  
 @Test  
 fun insertAndGetLocalRecords() = *runBlocking* **{** val repository = LeaderboardRepository(ApplicationProvider.getApplicationContext())  
  
 // Вставляем запись  
 val record = ScoreRecord(playerName = "Test", score = 200, date = "2024-01-15")  
 repository.insertLocalRecord(record)  
  
 // Проверяем что можем получить записи  
 val records = repository.getLocalRecords().first()  
 *assert*(records.*isNotEmpty*())  
 **}** @Test  
 fun fetchRemoteRecords() = *runBlocking* **{** val repository = LeaderboardRepository(ApplicationProvider.getApplicationContext())  
  
 // Пробуем загрузить данные с сервера  
 val remoteRecords = repository.fetchTop5Remote()  
  
 // Проверяем что функция работает (не падает)  
 *assert*(remoteRecords != null)  
 **}**}

Проверяет работу репозитория с локальными данными через Room и загрузку данных с mock-сервера, включая преобразование DTO в доменные модели.

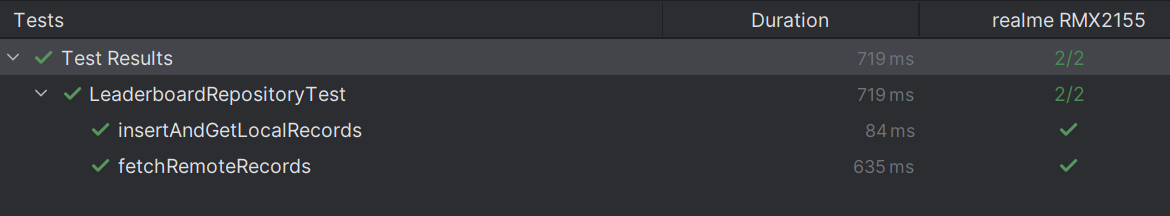


Рисунок 2. Успешно пройденный тест LeaderboardRepositoryTest

# Результат работы

Ссылка на GitHub: <https://github.com/AlekJulia/Alekseychik_MobileApplicationDevelopment.git>

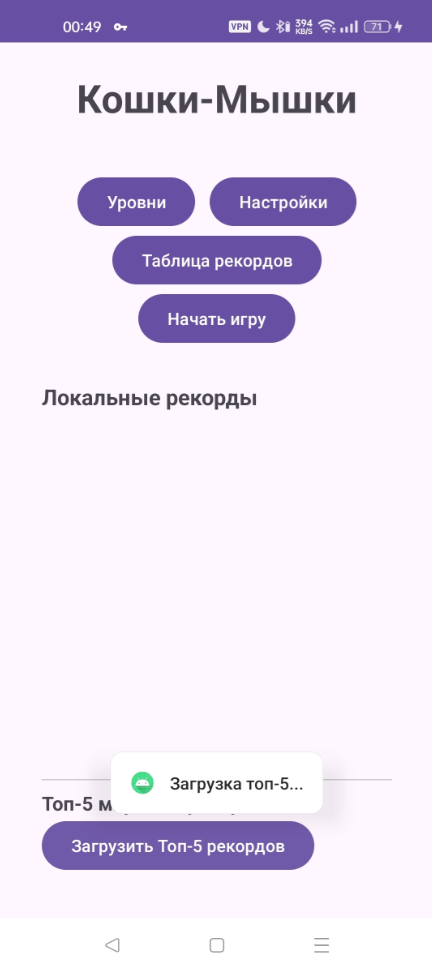


Рисунок 3. Загрузка рекордов с сервера

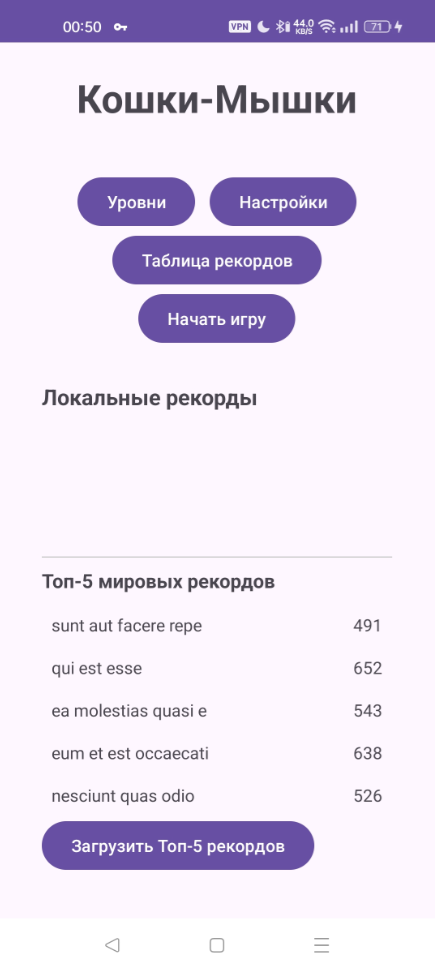


Рисунок 4. Загруженные рекорды.