Отчёт по практике

**«Разработка приложений для мобильных устройств»**

Задание 2

Реализация списка уровней и экрана настроек с использованием Fragment и ViewModel

Выполнил:

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

# Задание

Разработайте функционал выбора уровня сложности и экрана настроек игры.

1. Создайте класс данных Level (Уровень) со свойствами: id (идентификатор), name (название), difficulty (сложность, например, "Легкий", "Средний", "Сложный").
2. Реализуйте LevelsFragment, который будет отображать список уровней в RecyclerView. Каждый элемент списка должен отображать название уровня и его сложность.
3. Создайте SettingsFragment, который содержит:

* Переключатель (Switch) для включения/выключения фоновой музыки.
* Поле ввода (EditText) для имени игрока (значение по умолчанию - "Игрок").

1. Используйте SharedPreferences для сохранения настроек (имя игрока и состояние музыки).
2. Реализуйте SharedViewModel (наследник ViewModel), который будет хранить выбранный уровень (как LiveData или StateFlow) и обмениваться данными между фрагментами. При выборе уровня в списке, информация о выборе должна сохраняться в ViewModel.

# ****Исходные данные:****

1. Проектирование UI: Макеты fragment\_levels.xml (с RecyclerView), fragment\_settings.xml (с Switch и EditText), элемент списка item\_level.xml.
2. Программирование на Kotlin: Класс Level, LevelsAdapter для RecyclerView, SharedViewModel, SettingsFragment с логикой работы с SharedPreferences.
3. Работа с Android Components: Использование FragmentManager для навигации, наблюдение за LiveData/StateFlow во фрагментах.
4. Тестирование: Написать инструментальный тест (Espresso) для проверки отображения списка уровней и модульный тест для SharedViewModel.

# Ход работы

1. Создайте класс данных Level (Уровень) со свойствами: id (идентификатор), name (название), difficulty (сложность, например, "Легкий", "Средний", "Сложный").

Level.kt

// Модель данных уровня игры  
data class Level(  
 val id: Int, // уникальный идентификатор уровня  
 val name: String, // название уровня  
 val difficulty: String // сложность: легкий, средний, сложный  
)

1. Реализуйте LevelsFragment, который будет отображать список уровней в RecyclerView. Каждый элемент списка должен отображать название уровня и его сложность.

RecyclerView – компонент Android для эффективного отображения больших наборов данных в прокручиваемом списке, который повторно использует элементы списка recycle.

LevelsAdapter.kt

// Адаптер для RecyclerView списка уровней  
class LevelsAdapter(  
 private val items: List<Level>, // список данных для отображения  
 private val onItemClick: (Level) -> Unit // колбэк функция при клике на элемент  
) : RecyclerView.Adapter<LevelsAdapter.LevelViewHolder>() { // адаптер наследуется от RecyclerView.Adapter  
  
 // вложенный класс, который содержит ссылки на элементы ui  
 inner class LevelViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {  
 val tvName: TextView = itemView.findViewById(R.id.*tvLevelName*)  
 val tvDifficulty: TextView = itemView.findViewById(R.id.*tvLevelDifficulty*)  
 }  
  
 // Создает новые экземпляры ViewHolder (элемент списка)  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): LevelViewHolder {  
 val view = LayoutInflater.from(parent.*context*)  
 .inflate(R.layout.*item\_level*, parent, false)  
 return LevelViewHolder(view)  
 }  
  
 // RecyclerView вызывает метод onBindViewHolder, чтобы обновить содержимое ViewHolder для определенной позиции  
 override fun onBindViewHolder(holder: LevelViewHolder, position: Int) {  
 // получение объекта Level для текущей позиции в списке данных items  
 val level = items[position]  
  
 // заполнение TextViews в ViewHolder данными из объекта Level  
 holder.tvName.*text* = level.name  
 holder.tvDifficulty.*text* = level.difficulty  
  
 // при клике вызывается переданный колбэк  
 holder.itemView.setOnClickListener **{** onItemClick(level)  
 **}** }  
  
 // возвращение общего количества элементов в списке  
 // RecyclerView использует это, чтобы знать, сколько строк ему нужно отобразить  
 override fun getItemCount(): Int = items.size  
}

Файл LevelsAdapter.kt управляет отображением данных в RecyclerView. В нём создаются и управляются элементы списка уровней, связываются данные уровня Level с ViewHolder, обрабатывается нажатие на элемент списка. ViewHolder – паттерн проектирования в Android, который хранит ссылки на View-компоненты элемента списка RecyclerView для их повторного использования.

LevelsFragment.kt

// фрагмент со списком уровней  
class LevelsFragment : Fragment() {  
  
 private lateinit var sharedViewModel: SharedViewModel  
  
 // список уровней  
 private val sampleLevels = *listOf*(  
 Level(1, "Уровень 1", "Легкий"),  
 Level(2, "Уровень 2", "Средний"),  
 Level(3, "Уровень 3", "Сложный"),  
 )  
  
 // создание ui для фрагмента  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? { // на основе fragment\_levels.xml  
 return inflater.inflate(R.layout.*fragment\_levels*, container, false)  
 }  
  
 // инициализация ui элементов  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 // получение общего ViewModel через активность (чтобы он был один на всё activity)  
 sharedViewModel = ViewModelProvider(requireActivity()).get(SharedViewModel::class.*java*)  
  
 val rv: RecyclerView = view.findViewById(R.id.*rvLevels*)  
 rv.*layoutManager* = LinearLayoutManager(requireContext())  
  
 val adapter = LevelsAdapter(sampleLevels) **{** level **->** // при выборе уровня сохраняем в ViewModel  
 sharedViewModel.selectLevel(level)  
  
 // сообщение пользователю об уровне  
 Toast.makeText(requireContext(), "Выбран ${level.name} (${level.difficulty})", Toast.*LENGTH\_SHORT*).show()  
 **}** rv.*adapter* = adapter  
 }  
}

Фрагмент LevelsFragment.kt используется для отображения экрана выбора уровней. В нём инициализируется RecyclerView с адаптером, показываются Toast уведомления при выборе уровня. Для передачи данных выбранного уровня используется SharedViewModel.

fragment\_levels.xml

<?xml version="1.0" encoding="utf-8"?>  
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:id="@+id/levels\_root"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:padding="16dp">  
  
 <androidx.recyclerview.widget.RecyclerView  
 android:id="@+id/rvLevels"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent" />  
  
</FrameLayout>

Контейнер для списка уровней, содержаний элемент RecyclerView.

Для шаблона элемента списка уровней используется файл item\_level.xml

1. Создайте SettingsFragment, который содержит:

* Переключатель (Switch) для включения/выключения фоновой музыки.
* Поле ввода (EditText) для имени игрока (значение по умолчанию - "Игрок").

1. Используйте SharedPreferences для сохранения настроек (имя игрока и состояние музыки).

SettingsFragment.kt

// фрагмент настроек: сохраняет имя игрока и состояние музыки в SharedPreferences  
class SettingsFragment : Fragment() {  
  
 // элементы ui  
 private lateinit var switchMusic: Switch  
 private lateinit var etPlayerName: EditText  
  
 // константы для SharedPreferences  
 private val PREFS\_NAME = "game\_prefs"  
 private val KEY\_MUSIC = "pref\_music"  
 private val KEY\_PLAYER\_NAME = "pref\_player\_name"  
 private val DEFAULT\_PLAYER\_NAME = "Игрок"  
  
 private lateinit var sharedViewModel: SharedViewModel  
  
 override fun onCreateView(  
 inflater: LayoutInflater, container: ViewGroup?,  
 savedInstanceState: Bundle?  
 ): View? {  
 // создание ui из fragment\_settings.xml  
 return inflater.inflate(R.layout.*fragment\_settings*, container, false)  
 }  
  
 override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  
 super.onViewCreated(view, savedInstanceState)  
  
 sharedViewModel = ViewModelProvider(requireActivity()).get(SharedViewModel::class.*java*)  
  
 // инициализация ui элементов, найденных в fragment\_settings.xml  
 switchMusic = view.findViewById(R.id.*switchMusic*)  
 etPlayerName = view.findViewById(R.id.*etPlayerName*)  
  
 // загрузка сохранённых настроек из SharedPreferences  
 val prefs = requireActivity().getSharedPreferences(PREFS\_NAME, Context.*MODE\_PRIVATE*)  
 val musicOn = prefs.getBoolean(KEY\_MUSIC, true)  
 val playerName = prefs.getString(KEY\_PLAYER\_NAME, DEFAULT\_PLAYER\_NAME) ?: DEFAULT\_PLAYER\_NAME  
  
 // приминение настроек к ui элементам  
 switchMusic.*isChecked* = musicOn  
 etPlayerName.setText(playerName)  
  
 // обработчик для переключателя музыки  
 switchMusic.setOnCheckedChangeListener **{** \_, isChecked **->** prefs.edit().putBoolean(KEY\_MUSIC, isChecked).apply()  
 **}** // сохранение имени при потере фокуса  
 etPlayerName.setOnFocusChangeListener **{** \_, hasFocus **->** if (!hasFocus) {  
 savePlayerNameIfNeeded()  
 }  
 **}** // сохнранение при уходе с фрагмента  
 *viewLifecycleOwner*.lifecycle.addObserver(androidx.lifecycle.LifecycleEventObserver **{** \_, event **->** if (event == androidx.lifecycle.Lifecycle.Event.*ON\_PAUSE*) {  
 savePlayerNameIfNeeded()  
 }  
 **}**)  
  
 }  
  
 // функция для сохранения имени игрока  
 private fun savePlayerNameIfNeeded() {  
 val name = etPlayerName.*text*.toString().*ifBlank* **{** DEFAULT\_PLAYER\_NAME **}** val prefs = requireActivity().getSharedPreferences(PREFS\_NAME, Context.*MODE\_PRIVATE*)  
 prefs.edit().putString(KEY\_PLAYER\_NAME, name).apply()  
 }  
}

Фрагмент SettingsFragmen.kt используется для управления настройками игры и сохраняет их в постоянное хранилище SharedPreferences. SharedPreferences – встроенный механизм Android для постоянного хранения простых данных в формате ключ-значение, обеспечивающий сохранность информации между сессиями работы приложения. Пользовательский интерфейс экрана настроек описан в файле разметки fragment\_settings.xml. В нём находятся основные элементы: переключатель Switch – для управления музыкой и поле EditText – для ввода имени игрока.

1. Реализуйте SharedViewModel (наследник ViewModel), который будет хранить выбранный уровень (как LiveData или StateFlow) и обмениваться данными между фрагментами. При выборе уровня в списке, информация о выборе должна сохраняться в ViewModel.

SharedViewModel.kt

// ViewModel для обмена выбранным уровнем между фрагментами  
class SharedViewModel : ViewModel() {  
  
 // внутренний MutableLiveData для выбранного уровня  
 private val \_selectedLevel = MutableLiveData<Level?>() // создаём элемент MutableLiveData который будет хранить даннные уровня Level  
  
 // внешний LiveData для наблюдения из фрагментов  
 val selectedLevel: LiveData<Level?> = \_selectedLevel  
  
 // метод для установки выбранного уровня  
 fun selectLevel(level: Level) {  
 \_selectedLevel.value = level  
 }  
  
 // метод для сброса выбранного уровня  
 fun clearSelection() {  
 \_selectedLevel.value = null  
 }  
}

SharedViewModel используется фрагментами для обмена данными на основе LiveData в пределах одной активности. LiveData – класс-контейнер, который позволяет наблюдать за изменениями данных. MutableLiveData – именяемая версия LiveData, которая предоставляет методы для изменения хранимых данных. Фрагменты приложения, выступают в роли подписчиков, регистрируются для наблюдения за LiveData и автоматически получают уведомления при любом изменении данных, что гарантирует согласованность состояния между различными экранами приложения.

Тестирование:

SharedViewModelTest.kt

Юнит-тест проверяет корректность работы метода selectLevel() в классе SharedViewModel.

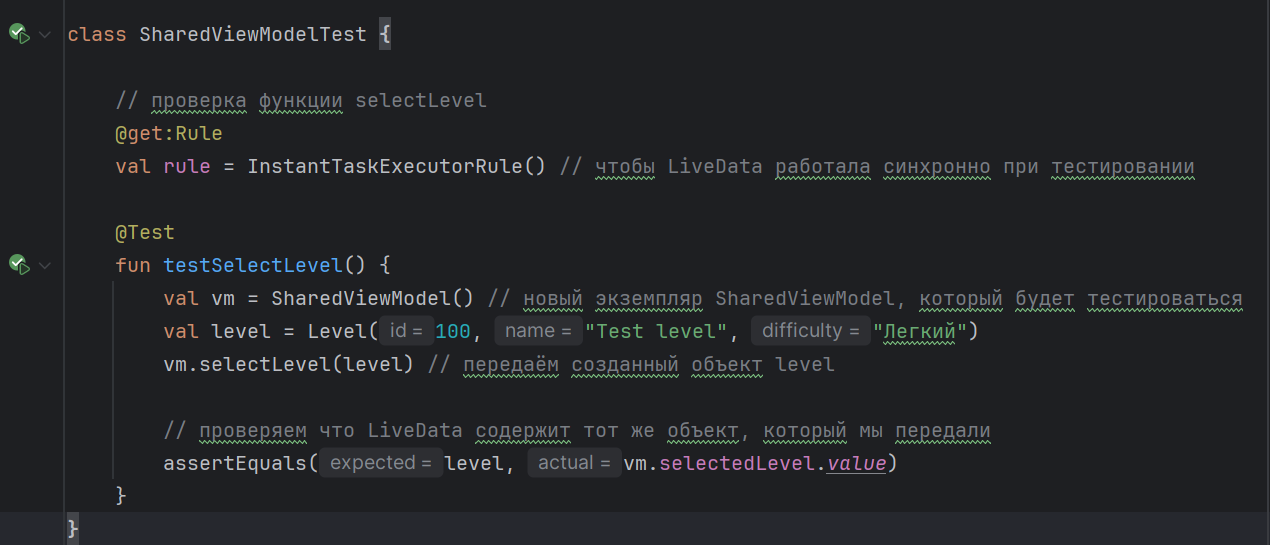


Рисунок 1. Успешное прохождение юнит-тестов.

LevelsFragmetTest.kt

@RunWith(AndroidJUnit4::class)  
class LevelsFragmentTest {  
 // проверка, что при нажатии на кнопку "уровни" открывается список уровней и на элемент списка можно кликнуть  
 @Test  
 fun levelsListIsDisplayed\_and\_itemClickable() {  
 // запуск MainActivity  
 ActivityScenario.launch(MainActivity::class.*java*).*use* **{** scenario **->** // нажатие на кнопку уровни  
 onView(withId(R.id.*btnLevels*)).perform(click())  
  
 // кликаем на элемент с уровнем  
 onView(withId(R.id.*rvLevels*))  
 .perform(RecyclerViewActions.actionOnItem<LevelsAdapter.LevelViewHolder>(  
 hasDescendant(withText("Уровень 2")), click()))  
 **}** }  
}

Для проверки работы главного экрана приложения используется espresso тест. Espresso – это фреймворк, который позволяет писать тесты, которые автоматически взаимодействуют с UI точно так же, как это делал бы реальный пользователь. Тест LevelsFragmetTest проверяет, может ли пользователь, нажав на кнопку “уровни”, взаимодействовать со списком уровней.

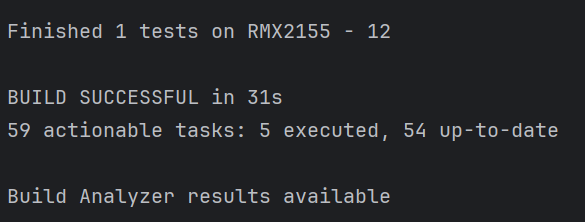


Рисунок 2. Успешное прохождение espresso теста.

# Результат работы

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

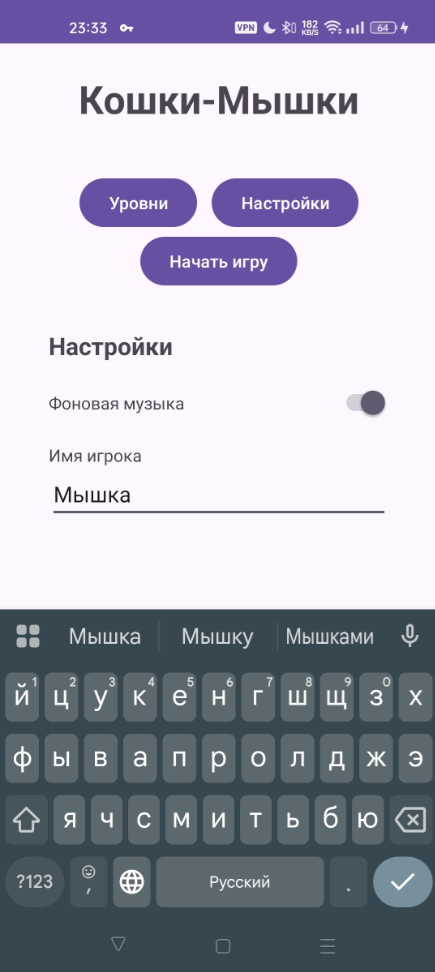


Рисунок 3. Настройки игры

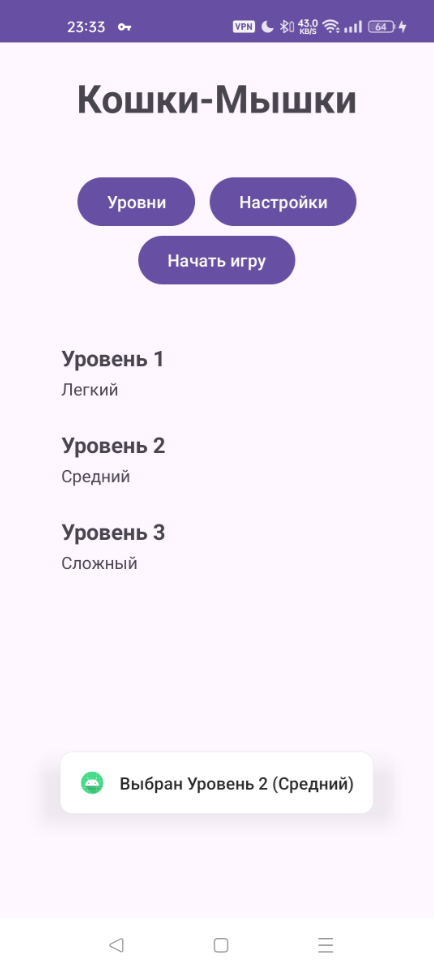


Рисунок 4. Выбор уровня