

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**"МИРЭА - Российский технологический университет"**

РТУ МИРЭА

Институт информационных технологий (ИТ)

Кафедра математического обеспечения и стандартизации информационных технологий (МОСИТ)

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2**

**по дисциплине**

**«Разработка клиент-серверных мобильных приложений»**

Выполнил студент группы ИКБО-11-22 Гришин А.В.

Принял к.э.н., доцент Исабекова О.А.

Москва 2025

# Задание 1

# На поисковой строке реализовать кнопку «Очистить».

# Обязательные условия:

# В поле ввода поискового запроса отображается подсказка, если поле ввода пустое.

# При нажатии на поле ввода появляется клавиатура.

# Если в поле ввода поискового запроса введён какой-то текст, то отображается кнопка «Очистить».

# Если в поле ввода поискового запроса нет никакого текста, то кнопка «Очистить» не отображается.

# Нажатие на кнопку «Очистить» удаляет текст в поле ввода поискового запроса и скрывает клавиатуру.

# 

# Рисунок 1 – Реализация поисковой строки

# В поле ввода поискового запроса была добавлена подсказка, которая отображается, если поле ввода пустое. Кнопка очистки видна только при наличии текста в поле. При нажатии на поле ввода появляется клавиатура для ввода текста.

# Задание 2

# Реализовать сохранение текста поискового запроса в жизненном цикле, то есть если в поле ввода поискового запроса есть какой-то текст, то при повороте устройства в ландшафтную ориентацию и обратно текст в поле ввода поискового запроса не исчезает.

# 

# Рисунок 2 – Сохранение текста при перевороте экрана

# Для сохранения текста в поисковой строке при повороте устройства было реализовано сохранение состояния поискового запроса в жизненном цикле приложения. Это позволяет сохранить введённый текст при переходе из портретной ориентации в ландшафтную и обратно.

# Задание 3

# Если в приложении для курсовой работы не предполагается подключение API, необходимо подготовить «приложение-заглушку» для демонстрации работы API.

# Поскольку курсовая работа предполагает подключение API, необходимость в «приложении-заглушке» отпала. Вместо этого было решено подключить реальную API для демонстрации функциональности. Использованные классы: GameApiService, RetrofitClient, GameAdapter, Game, GameResponse.

# Задание 4

# Подключить к странице поиска любую открытую API. Ссылку на список открытых API можно найти в СДО. Реализовать корректный поиск, согласно теме, проект.

# 

# Рисунок 3 – Результат поиска согласно теме

# Для реализации функциональности поиска в мобильном приложении был подключён внешний API — RAWG Video Games Database API, предоставляющий обширную базу данных видеоигр. С его помощью реализован механизм отправки HTTP-запросов с параметром поиска, в ответ на который сервер возвращает JSON-объект со списком игр, соответствующих введённому запросу.

# Интерфейс Api:

# getGames:

# Описание: Этот метод выполняет HTTP GET-запрос к эндпоинту games публичного RAWG API. Используется для получения списка игр, соответствующих поисковому запросу пользователя.

# • Параметры:

# query: Строка, представляющая текст поискового запроса, введённый пользователем (используется параметр search в URL).

# apiKey: Строка с API-ключом, необходимая для аутентификации запроса (параметр key).

# pageSize: Целое число, указывающее количество результатов на странице. По умолчанию установлено значение 20.

# • Возвращаемое значение:

# Call<GameResponse> — объект, асинхронно возвращающий ответ от сервера, содержащий список найденных игр.

# Класс GameAdapter

# updateGames:

# Описание: Обновляет текущий список игр новым и перерисовывает RecyclerView.

# Параметры:

# newGames: Новый список объектов Game.

# Реализация:

# Перезаписывает список и вызывает notifyDataSetChanged().

# Задание 5

# Реализовать плейсхолдеры:

# Если нет результатов поиска.

# Если поисковый запрос завершился неудачно, при этом на данном плейсхолдере должна быть размещена кнопка «Обновить».

# 

# Рисунок 4 – Появление кнопки “Обновить” при неудачном результате поиска

# Были реализованы плейсхолдеры для различных сценариев:

# Отсутствие результатов поиска: Если поиск не дал результатов, отображается соответствующее сообщение («Нет результатов»).

# Неудачный поисковый запрос: В случае неудачного запроса отображается плейсхолдер «Ошибка загрузки данных» с кнопкой «Обновить».

# Задание 6

# Если на плейсхолдере присутствует кнопка «Обновить», то необходимо реализовать повторное отправление последнего запроса при нажатии на нее.

Была реализована функциональность кнопки «Обновить», предназначенной для повторной отправки последнего поискового запроса. При возникновении ошибки в процессе загрузки данных или при отсутствии подключения к сети пользователь видит соответствующий плейсхолдер с данной кнопкой. Нажатие на неё автоматически инициирует повторную попытку выполнить последний запрос без необходимости повторного ввода текста вручную. Это повышает удобство использования приложения и улучшает пользовательский опыт за счёт быстрой возможности повторного получения результатов.

*Листинг 1 – Код GameApiService.kt*

|  |
| --- |
| package com.rarmash.b4cklog.network  import com.rarmash.b4cklog.models.GameResponse  import retrofit2.Call  import retrofit2.http.GET  import retrofit2.http.Query  interface GameApiService {  @GET("games")  fun getGames(  @Query("search") query: String,  @Query("key") apiKey: String,  @Query("page\_size") pageSize: Int = 20  ): Call<GameResponse>  } |

*Листинг 2 – Код RetrofitClient.kt*

|  |
| --- |
| package com.rarmash.b4cklog.network  import okhttp3.OkHttpClient  import okhttp3.logging.HttpLoggingInterceptor  import retrofit2.Retrofit  import retrofit2.converter.gson.GsonConverterFactory  object RetrofitClient {  private const val BASE\_URL = "https://api.rawg.io/api/"  const val API\_KEY = "404f3f12c8f2494d87216668b5be8963"  private val logging = HttpLoggingInterceptor().apply {  level = HttpLoggingInterceptor.Level.BODY  }  private val client = OkHttpClient.Builder()  .addInterceptor(logging)  .build()  private val retrofit = Retrofit.Builder()  .baseUrl(BASE\_URL)  .addConverterFactory(GsonConverterFactory.create())  .client(client)  .build()  val gameApiService: GameApiService = retrofit.create(GameApiService::class.java)  } |

*Листинг 3 – Код GameAdapter.kt*

|  |
| --- |
| package com.rarmash.b4cklog.adapters  import android.view.LayoutInflater  import android.view.View  import android.view.ViewGroup  import android.widget.ImageView  import android.widget.TextView  import androidx.recyclerview.widget.RecyclerView  import com.bumptech.glide.Glide  import com.rarmash.b4cklog.R  import com.rarmash.b4cklog.models.Game  class GameAdapter(private var games: List<Game>) : RecyclerView.Adapter<GameAdapter.GameViewHolder>() {  class GameViewHolder(view: View) : RecyclerView.ViewHolder(view) {  val gameName: TextView = view.findViewById(R.id.gameName)  val gameCover: ImageView = view.findViewById(R.id.gameCover)  }  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): GameViewHolder {  val view = LayoutInflater.from(parent.context).inflate(R.layout.item\_game, parent, false)  return GameViewHolder(view)  }  override fun onBindViewHolder(holder: GameViewHolder, position: Int) {  val game = games[position]  holder.gameName.text = game.name  Glide.with(holder.itemView.context)  .load(game.imageUrl ?: R.drawable.placeholder\_image)  .placeholder(R.drawable.placeholder\_image)  .into(holder.gameCover)  }  override fun getItemCount(): Int = games.size  fun updateGames(newGames: List<Game>) {  games = newGames  notifyDataSetChanged()  }  } |

*Листинг 4 – Код Game.kt*

|  |
| --- |
| package com.rarmash.b4cklog.models  import com.google.gson.annotations.SerializedName  data class Game(  @SerializedName("id") val id: Int,  @SerializedName("name") val name: String,  @SerializedName("released") val releaseDate: String?,  @SerializedName("background\_image") val imageUrl: String?  ) |

*Листинг 5 – Код GameResponse.kt*

|  |
| --- |
| package com.rarmash.b4cklog.models  import com.google.gson.annotations.SerializedName  data class GameResponse(  @SerializedName("results") val games: List<Game>  ) |

*Листинг 6 – Код SearchFragment.kt*

|  |
| --- |
| package com.rarmash.b4cklog.fragments  import android.os.Bundle  import android.util.Log  import android.view.LayoutInflater  import android.view.View  import android.view.ViewGroup  import android.view.inputmethod.InputMethodManager  import android.content.Context  import android.graphics.Color  import android.os.Handler  import android.os.Looper  import android.widget.EditText  import android.widget.Toast  import androidx.fragment.app.Fragment  import androidx.recyclerview.widget.LinearLayoutManager  import com.rarmash.b4cklog.adapters.GameAdapter  import com.rarmash.b4cklog.databinding.FragmentSearchBinding  import com.rarmash.b4cklog.models.GameResponse  import com.rarmash.b4cklog.network.RetrofitClient  import com.rarmash.b4cklog.network.RetrofitClient.API\_KEY  import retrofit2.Call  import retrofit2.Callback  import retrofit2.Response  class SearchFragment : Fragment() {  private lateinit var binding: FragmentSearchBinding  private lateinit var gameAdapter: GameAdapter  private var lastQuery: String? = null  private val handler = Handler(Looper.getMainLooper())  private var searchRunnable: Runnable? = null  override fun onCreateView(  inflater: LayoutInflater,  container: ViewGroup?,  savedInstanceState: Bundle?  ): View {  binding = FragmentSearchBinding.inflate(inflater, container, false)  gameAdapter = GameAdapter(emptyList())  binding.recyclerView.layoutManager = LinearLayoutManager(context)  binding.recyclerView.adapter = gameAdapter  binding.searchView.clearFocus()  binding.searchView.findViewById<EditText>(androidx.appcompat.R.id.search\_src\_text).setHintTextColor(Color.DKGRAY);  binding.searchView.queryHint = "Искать игру"  savedInstanceState?.let {  lastQuery = it.getString("lastQuery")  binding.searchView.setQuery(lastQuery, false)  lastQuery?.let {  fetchGames(it)  }  }  return binding.root  }  override fun onViewCreated(view: View, savedInstanceState: Bundle?) {  super.onViewCreated(view, savedInstanceState)  binding.searchView.setOnQueryTextListener(object : androidx.appcompat.widget.SearchView.OnQueryTextListener {  override fun onQueryTextSubmit(query: String?): Boolean {  return false  }  override fun onQueryTextChange(newText: String?): Boolean {  searchRunnable?.let { handler.removeCallbacks(it) }  searchRunnable = Runnable {  newText?.let {  lastQuery = it  fetchGames(it)  }  }  if (newText.isNullOrEmpty()) {  binding.searchView.clearFocus() // Убираем фокус при очистке  binding.searchView.postDelayed({  hideKeyboard()  }, 50)  }  handler.postDelayed(searchRunnable!!, 500) // Задержка 500 мс перед выполнением запроса  return false  }  })  binding.retryButton.setOnClickListener {  lastQuery?.let { fetchGames(it) }  }  }  private fun fetchGames(searchQuery: String) {  showLoadingState()  RetrofitClient.gameApiService.getGames(searchQuery, API\_KEY).enqueue(object : Callback<GameResponse> {  override fun onResponse(call: Call<GameResponse>, response: Response<GameResponse>) {  Log.d("SearchFragment", "Код ответа: ${response.code()}")  if (response.isSuccessful) {  val games = response.body()?.games ?: emptyList()  gameAdapter.updateGames(games)  updatePlaceholder(games.isEmpty())  } else {  Log.e("SearchFragment", "Ошибка: ${response.code()} ${response.errorBody()?.string() ?: "Нет инфо"}")  showErrorPlaceholder()  }  }  override fun onFailure(call: Call<GameResponse>, t: Throwable) {  Toast.makeText(context, "Ошибка сети: ${t.message}", Toast.LENGTH\_SHORT).show()  showErrorPlaceholder()  }  })  }  private fun hideKeyboard() {  val imm = requireContext().getSystemService(Context.INPUT\_METHOD\_SERVICE) as InputMethodManager  imm.hideSoftInputFromWindow(binding.searchView.windowToken, 0)  }  private fun showLoadingState() {  binding.recyclerView.visibility = View.GONE  binding.placeholderLayout.visibility = View.GONE  }  private fun updatePlaceholder(isEmpty: Boolean) {  binding.recyclerView.visibility = if (isEmpty) View.GONE else View.VISIBLE  binding.placeholderLayout.visibility = if (isEmpty) View.VISIBLE else View.GONE  binding.placeholderText.text = "Нет результатов"  binding.retryButton.visibility = View.GONE  }  private fun showErrorPlaceholder() {  binding.recyclerView.visibility = View.GONE  binding.placeholderLayout.visibility = View.VISIBLE  binding.placeholderText.text = "Ошибка загрузки данных"  binding.retryButton.visibility = View.VISIBLE  }  override fun onSaveInstanceState(outState: Bundle) {  super.onSaveInstanceState(outState)  outState.putString("lastQuery", lastQuery)  }  } |

# Вывод

В ходе выполнения практической работы были успешно реализованы все поставленные задачи, связанные с разработкой клиент-серверного мобильного приложения «Игровой бэклог». Были созданы и протестированы основные экраны приложения, включая экран поиска с поддержкой очистки строки ввода, сохранением текста при повороте экрана, отображением истории запросов и автоматическим выполнением поиска.

Также была реализована интеграция с внешним API RAWG, что позволило получать актуальные данные об играх на основе пользовательских запросов. Обработка возможных ошибок и отображение плейсхолдеров (например, при отсутствии результатов или сбоях сети) значительно повысили удобство взаимодействия с приложением. Кнопка «Обновить» дала пользователю возможность повторно отправлять последний запрос без необходимости ручного ввода.

В результате было разработано функциональное и удобное мобильное приложение, которое позволяет пользователям эффективно искать и отслеживать интересующие их видеоигры.

# Список используемой литературы

1. Сомон., П. И. Волшебство Kotlin : руководство / П. И. Сомон. ; перевод с английского А. Н. Киселева.. — Москва : ДМК Пресс, 2020. — 536 с. — ISBN 978-5-97060-801-2. — Текст : электронный // Лань : электронно- библиотечная система. — URL: https://e.lanbook.com/book/140599 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.
2. Калгина, И. С. Разработка мобильных приложений : учебное пособие / И. С. Калгина. — Чита : ЗабГУ, 2022. — 163 с. — ISBN 978-5-9293-3137-4. —

Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/363323 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.

1. Алпатов, А. Н. Архитектура, проектирование и разработка программных средств : учебное пособие / А. Н. Алпатов, И. Е. Рогов. — Москва : РТУ МИРЭА, 2023. — 120 с. — ISBN 978-5-7339-1972-0. — Текст : электронный

// Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/386189 (дата обращения: 15.02.2024). — Режим доступа: для авториз. пользователей.