**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное

учреждение высшего образования

**Национальный исследовательский**

**Томский политехнический университет**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Отчет по лабораторной работе №24 по дисциплине

**«Разработка приложений для мобильных устройств»**

Интернет и JSON

Выполнил:

Студент группы 8В21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Алексейчик Ю. Д.

Проверил:

Ст. преп. ОИТ ИШИТР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дорофеев В. А.

# Задание

Разработайте программу-игру «Найди точку!» Программа должна использовать средства геолокации и должна выглядеть примерно так:

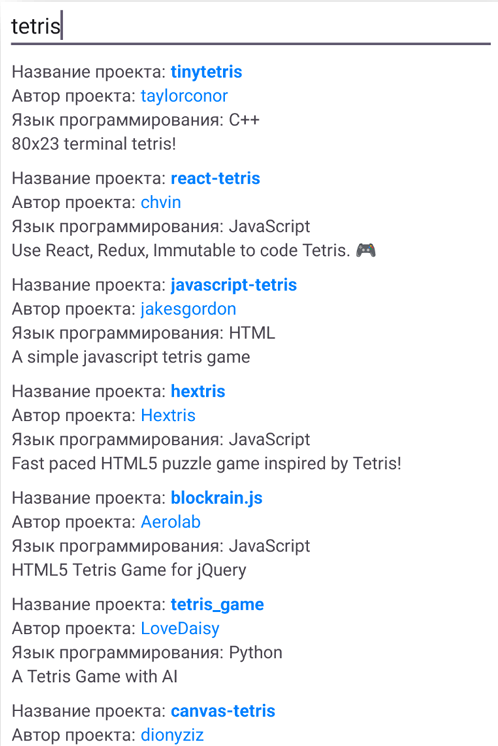


Рисунок 1. Примерный вид приложения

Название репозитория и логин владельца репозитория должны быть выделены другим цветом, при нажатии на них в браузере по умолчанию должна открываться страница проекта или страница владельца репозитория, соответственно.

# Ход работы

Для получаемых данных от GitHub используем data-классы, представляющие собой структуры данных.

**Owner.kt**

// модель владельца репозитория  
data class Owner(  
 val login: String, // логин владельца  
 val html\_url: String // URL страницы владельца  
)

**RepoItem.kt**

// модель репозитория  
data class RepoItem(  
 val name: String, // название репозитория  
 val owner: Owner, // владелец (объект Owner)  
 val html\_url: String, // URL репозитория  
 val description: String?, // описание   
 val language: String? // язык программирования   
)

**RepoResponse.kt**

// модель ответа от API GitHub  
data class RepoResponse(  
 val items: List<RepoItem> // список найденных репозиториев  
)

Для взаимодействия с GitHub API через HTTP-запросы используются файлы: GithubApi.kt – интерфейс для определения API-эндпоинтов, RetrofitInstance.kt – настройка Retrofit клиента.

**GithubApi.kt**

interface GithubApi {  
 // Запрос к: https://api.github.com/search/repositories?q=<запрос>  
 @GET("search/repositories")  
 suspend fun searchRepositories(  
 @Query("q") query: String  
 ): RepoResponse  
}

**RetrofitInstance.kt**

object RetrofitInstance {  
 private val logging = HttpLoggingInterceptor().*apply* **{** level = HttpLoggingInterceptor.Level.*BASIC* // логирование HTTP-запросов  
 **}** private val client = OkHttpClient.Builder()  
 .addInterceptor(logging) // добавление интерцептора для логирования  
 .build()  
  
 // создание экземпляра Retrofit с базовым URL и конвертером Gson  
 private val retrofit by *lazy* **{** Retrofit.Builder()  
 .baseUrl("https://api.github.com/") // базовый URL GitHub API  
 .addConverterFactory(GsonConverterFactory.create()) // конвертер JSON в объекты  
 .client(client)  
 .build()  
 **}** // инициализация API  
 val api: GithubApi by *lazy* **{** retrofit.create(GithubApi::class.*java*)  
 **}**}

**SearchViewModel.kt**

class SearchViewModel : ViewModel() {  
 // livedata для наблюдения за результатами поиска  
 val results = MutableLiveData<List<RepoItem>>()  
 // livedata для наблюдения за ошибками  
 val error = MutableLiveData<String?>()  
  
 // функция поиска репозиториев  
 fun search(query: String) {  
 *viewModelScope*.*launch* **{** // запуск корутины в scope viewmodel  
 try {  
 // выполнение сетевого запроса  
 val response = RetrofitInstance.api.searchRepositories(query)  
 // публикация результатов в livedata  
 results.postValue(response.items)  
 error.postValue(null) // очистка ошибок при успешном запросе  
 } catch (e: Exception) {  
 // обработка ошибок  
 results.postValue(*emptyList*())  
 error.postValue(e.*localizedMessage* ?: "ошибка запроса")  
 }  
 **}** }  
}

SearchViewModel.kt управляет бизнес-логикой приложения и предоставляет данные для UI.

Пользовательский интерфейс представлен в файле activity\_main.xml, который содержит:

* EditText для ввода поискового запроса
* Button для запуска поиска
* RecyclerView для отображения результатов

item\_repo.xml - элемент списка репозиториев, который отображает название репозитория, владельца и языка программирования.

Для представления репозиториев в виде списка используется адаптер RecyclerView, который позволяет открыть ссылки на репозитории в браузере при нажатии на элемент списка.

**RepoAdapter.kt**

class RepoAdapter(private val context: Context, private var list: List<RepoItem>) :  
 RecyclerView.Adapter<RepoAdapter.VH>() {  
  
 // viewholder для кэширования view-компонентов  
 inner class VH(itemView: View) : RecyclerView.ViewHolder(itemView) {  
 val tvName: TextView = itemView.findViewById(R.id.*tvName*)  
 val tvOwner: TextView = itemView.findViewById(R.id.*tvOwner*)  
 val tvLanguage: TextView = itemView.findViewById(R.id.*tvLanguage*)  
 }  
  
 override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): VH {  
 // создание нового viewholder при необходимости  
 val view = LayoutInflater.from(parent.*context*).inflate(R.layout.*item\_repo*, parent, false)  
 return VH(view)  
 }  
  
 override fun onBindViewHolder(holder: VH, position: Int) {  
 val repo = list[position]  
  
 // установка данных в view-компоненты  
 holder.tvName.*text* = context.getString(R.string.*repo\_name*, repo.name)  
 holder.tvOwner.*text* = context.getString(R.string.*repo\_owner*, repo.owner.login)  
 holder.tvLanguage.*text* = context.getString(R.string.*repo\_lang*, repo.language ?: "—")  
  
 // обработчики кликов для открытия url  
 holder.tvName.setOnClickListener **{** openUrl(repo.html\_url) // открытие репозитория  
 **}** holder.tvOwner.setOnClickListener **{** openUrl(repo.owner.html\_url) // открытие профиля владельца  
 **}** }  
  
 // обновление данных адаптера  
 fun updateData(newList: List<RepoItem>) {  
 list = newList  
 notifyDataSetChanged() // уведомление об изменении данных  
 }  
  
 // открытие url в браузере  
 private fun openUrl(url: String) {  
 val intent = Intent(Intent.*ACTION\_VIEW*, Uri.parse(url))  
 intent.addFlags(Intent.*FLAG\_ACTIVITY\_NEW\_TASK*)  
 context.startActivity(intent)  
 }  
  
 override fun getItemCount(): Int = list.size  
}

Основная логика инициализации UI и запуска поиска процесса репозитория находится в MainActivity.kt.

**MainActivity.kt**

class MainActivity : AppCompatActivity() {  
  
 // ViewModel — содержит всю логику запроса к API  
 private val viewModel: SearchViewModel by *viewModels*()  
  
 // Элементы интерфейса  
 private lateinit var etQuery: EditText  
 private lateinit var btnSearch: Button  
 private lateinit var rvResults: RecyclerView  
  
 // адаптер для списка репозиториев  
 private lateinit var adapter: RepoAdapter  
  
 override fun onCreate(savedInstanceState: Bundle?) {  
 super.onCreate(savedInstanceState)  
 setContentView(R.layout.*activity\_main*)  
  
 // инициализация UI-элементов  
 etQuery = findViewById(R.id.*etQuery*)  
 btnSearch = findViewById(R.id.*btnSearch*)  
 rvResults = findViewById(R.id.*rvResults*)  
  
 // инициализация адаптера (передаём пустой список в начале)  
 adapter = RepoAdapter(*applicationContext*, *emptyList*())  
  
 // Настраиваем RecyclerView — вертикальный список  
 rvResults.*layoutManager* = LinearLayoutManager(this)  
 rvResults.*adapter* = adapter  
  
 // кнопка поиск запускает поиск  
 btnSearch.setOnClickListener **{** val query = etQuery.*text*.toString().*trim*()  
 if (query.*isNotEmpty*()) {  
 doSearch(query)  
 } else {  
 Toast.makeText(this, getString(R.string.*hint\_search*), Toast.*LENGTH\_SHORT*).show()  
 }  
 **}** // Обработка нажатия клавиши поиск на клавиатуре  
 etQuery.setOnEditorActionListener **{** \_, actionId, \_ **->** if (actionId == EditorInfo.*IME\_ACTION\_SEARCH*) {  
 val query = etQuery.*text*.toString().*trim*()  
 if (query.*isNotEmpty*()) {  
 doSearch(query)  
 }  
 true  
 } else false  
 **}** // подписка на результаты из ViewModel  
 viewModel.results.observe(this) **{** list **->** onResultsReceived(list)  
 **}** // подписка на возможные ошибки  
 viewModel.error.observe(this) **{** errorMsg **->** errorMsg?.*let* **{** Toast.makeText(this, **it**, Toast.*LENGTH\_LONG*).show()  
 **}  
 }** }  
  
 private fun doSearch(query: String) {  
 viewModel.search(query)  
 Toast.makeText(this, getString(R.string.*btn\_search*) + ": " + query, Toast.*LENGTH\_SHORT*).show()  
 }  
  
  
 private fun onResultsReceived(list: List<RepoItem>) {  
 if (list.isEmpty()) {  
 Toast.makeText(this, getString(R.string.*no\_results*), Toast.*LENGTH\_SHORT*).show()  
 }  
 adapter.updateData(list)  
 }  
}

# Результаты работы

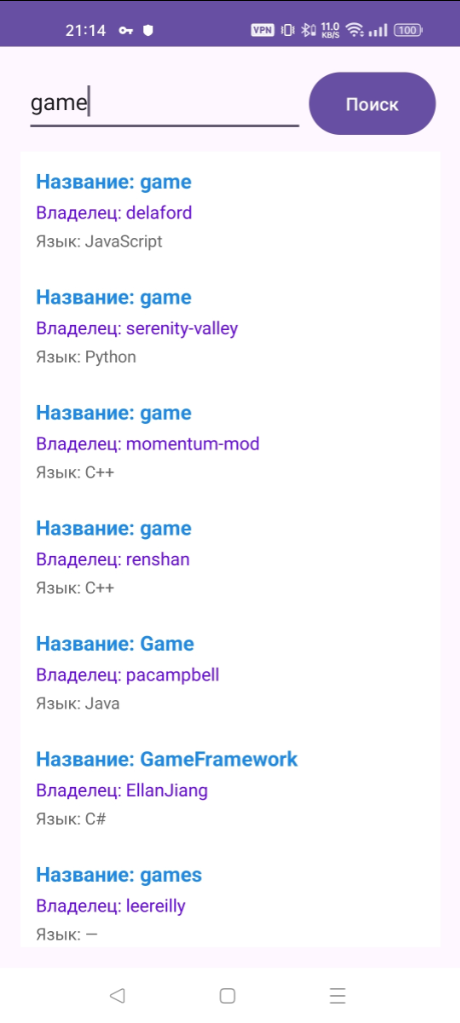


Рисунок 2. Поиск репозитория

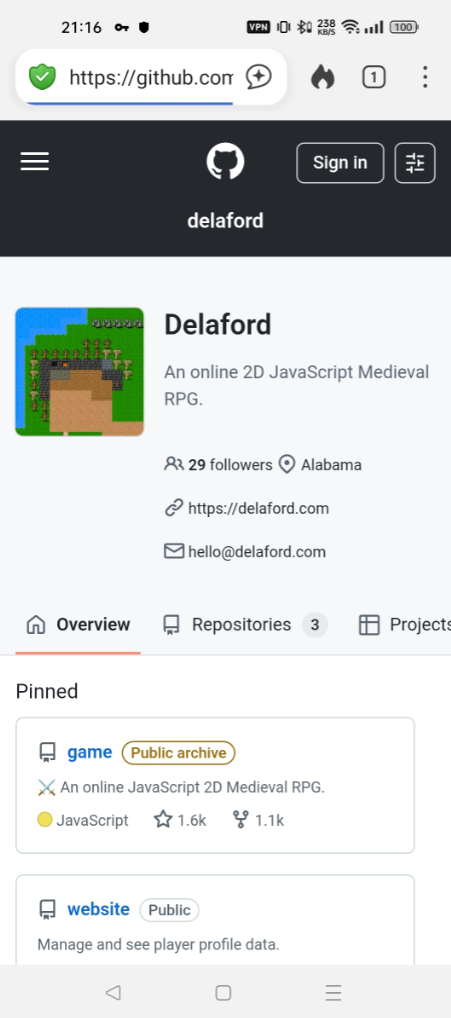


Рисунок 3. Переход по ссылке при нажатии на найденный репозиторий

# Выводы

В ходе выполнения лабораторной работы было разработано приложение для поиска репозиториев в GitHub. Была организована работа с сетевыми запросами с использованием библиотеки Retrofit, настроено отображение данных в RecyclerView с возможностью, при нажатии на название репозитория или имя владельца, открывать соответствующие веб-страницы в браузере.