

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 №3**

**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

**Практическая работа 3**

# Задание 1

Переключатель темной темы реализован посредством добавления элемента Switch на экране настроек (идентификатор themeSwitch). При нажатии на этот Switch происходит вызов логики, которая изменяет текущий режим темы приложения. Это делается через вызов метода из библиотеки поддержки с использованием AppCompatDelegate.setDefaultNightMode() (листинг 1, рис. 1, 2).

*Листинг 1 – Смена темы в приложении*

|  |
| --- |
| themeSwitch.setOnCheckedChangeListener { \_, isChecked ->  sharedPreferences.edit().putBoolean("dark\_mode", isChecked).apply()  AppCompatDelegate.setDefaultNightMode(  if (isChecked) AppCompatDelegate.MODE\_NIGHT\_YES  else AppCompatDelegate.MODE\_NIGHT\_NO  )  requireActivity().recreate()  } |

**

Рисунок 1 – Часть экрана со светлой темой

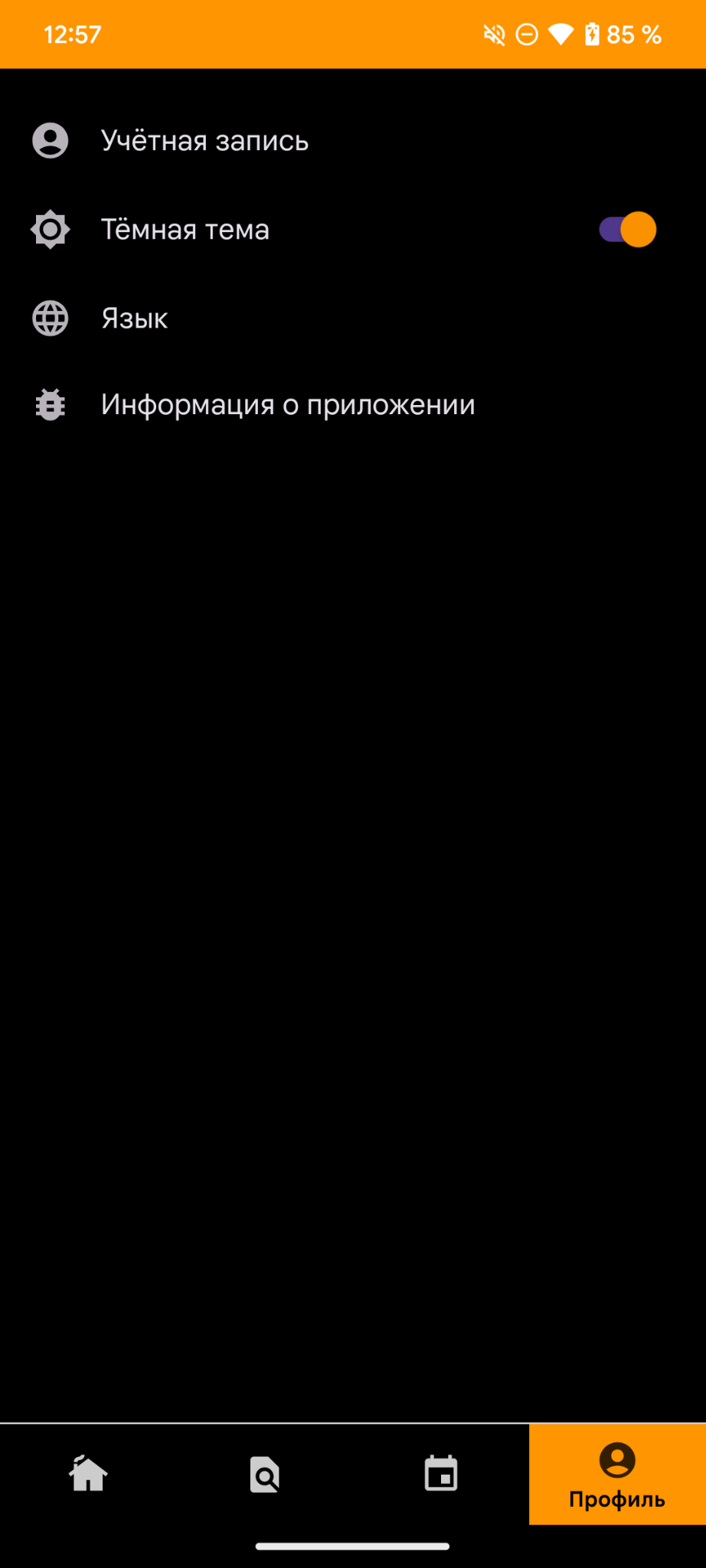


Рисунок 2 – Часть экрана с темной темой

# Задание 2. Хранение истории поиска с помощью SharedPreferences

Для хранения истории поисковых запросов используется механизм SharedPreferences. Основные моменты реализации:

1. Показ истории: при получении фокуса в поле ввода (класс androidx.appcompat.widget.SearchView) вызывается метод, который загружает историю из SharedPreferences. Если история не пуста, она отображается с помощью списка (листинг 2, 3, рис. 3).

*Листинг 2 – Сохранение запроса в историю поиска*

|  |
| --- |
| private fun saveToHistory(query: String) {  val prefs = requireContext().getSharedPreferences("search\_prefs", Context.MODE\_PRIVATE)  val history = loadHistory().toMutableList()  history.remove(query)  history.add(0, query)  val limited = history.take(MAX\_HISTORY\_SIZE)  val jsonArray = JSONArray()  for (item in limited) {  jsonArray.put(item)  }  prefs.edit().putString(HISTORY\_PREF, jsonArray.toString()).apply()  } |

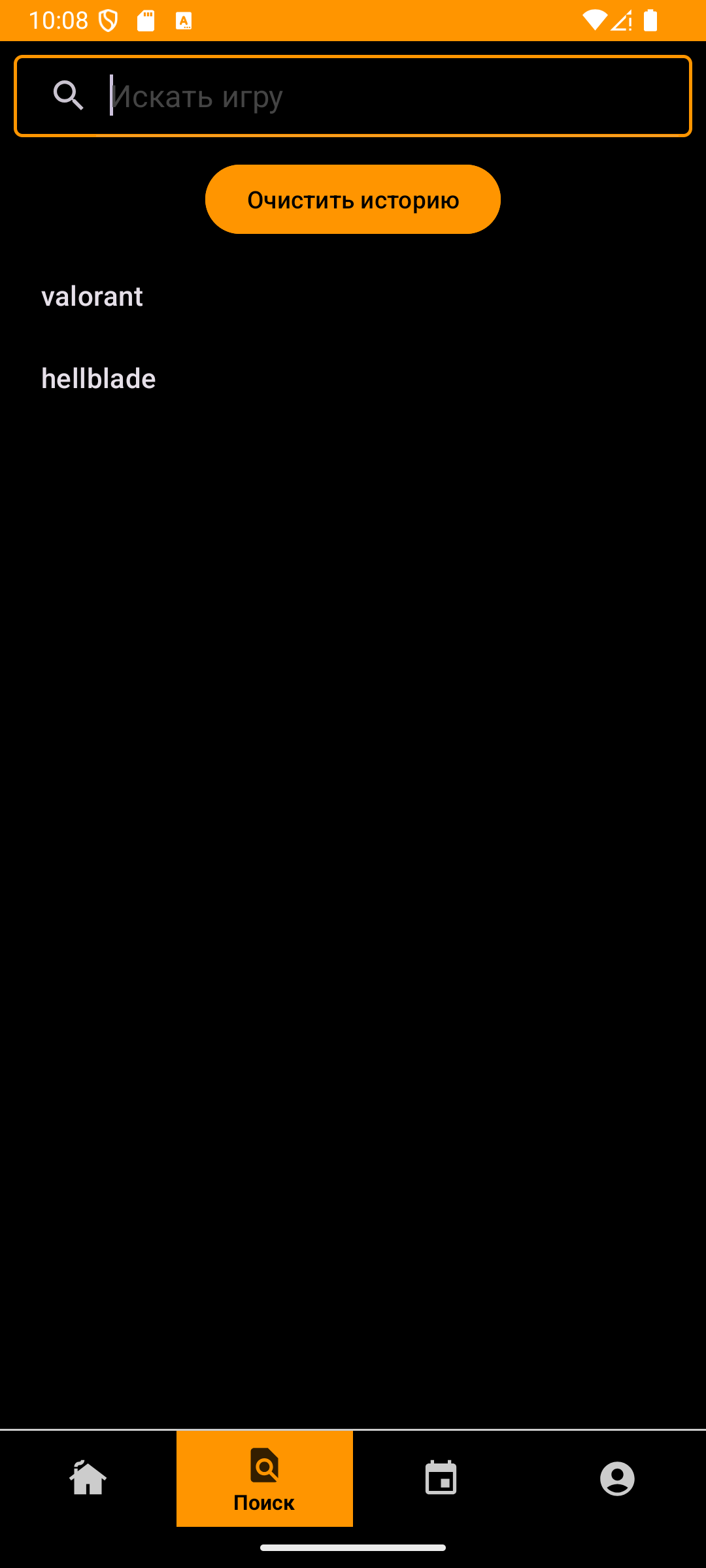
**

Рисунок 3 – Показ истории поиска

*Листинг 3 – Отображение истории (при фокусе в поле ввода):*

|  |
| --- |
| private fun showHistory() {  val history = loadHistory()  if (history.isNotEmpty()) {  isShowingHistory = true  binding.recyclerView.adapter = historyAdapter  historyAdapter.update(history)  binding.recyclerView.visibility = View.VISIBLE  binding.clearHistoryButton.visibility = View.VISIBLE  binding.placeholderLayout.visibility = View.GONE  }  } |

1. Ограничение по количеству: в истории хранится не более 10 элементов. Если число запросов превышает 10, из списка удаляются элементы, начиная с 11-го (листинг 4, рис. 4)

*Листинг 4 – Лимитирование количества сохранённых запросов*

|  |
| --- |
| history.add(0, query)  val limited = history.take(MAX\_HISTORY\_SIZE) // 10  val jsonArray = JSONArray()  for (item in limited) {  jsonArray.put(item)  } |

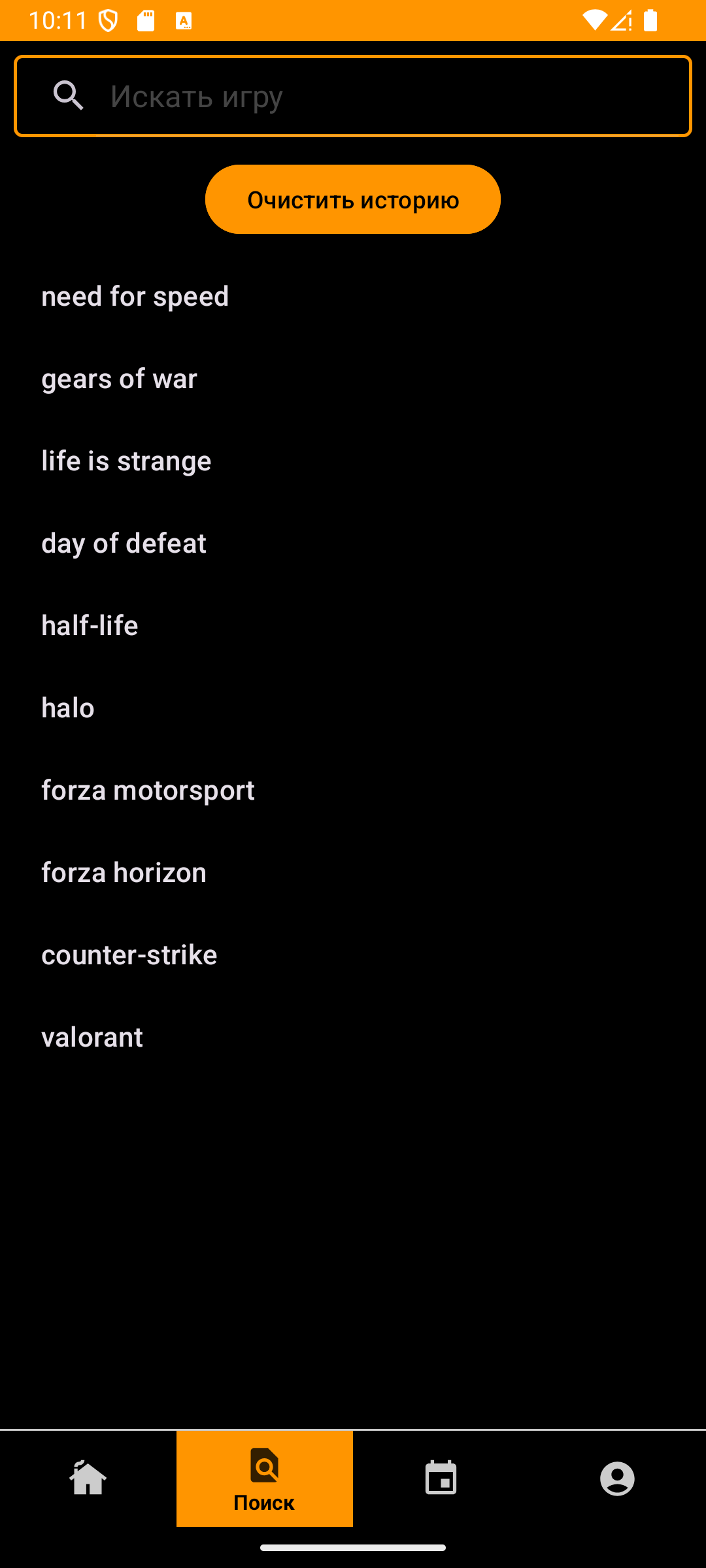
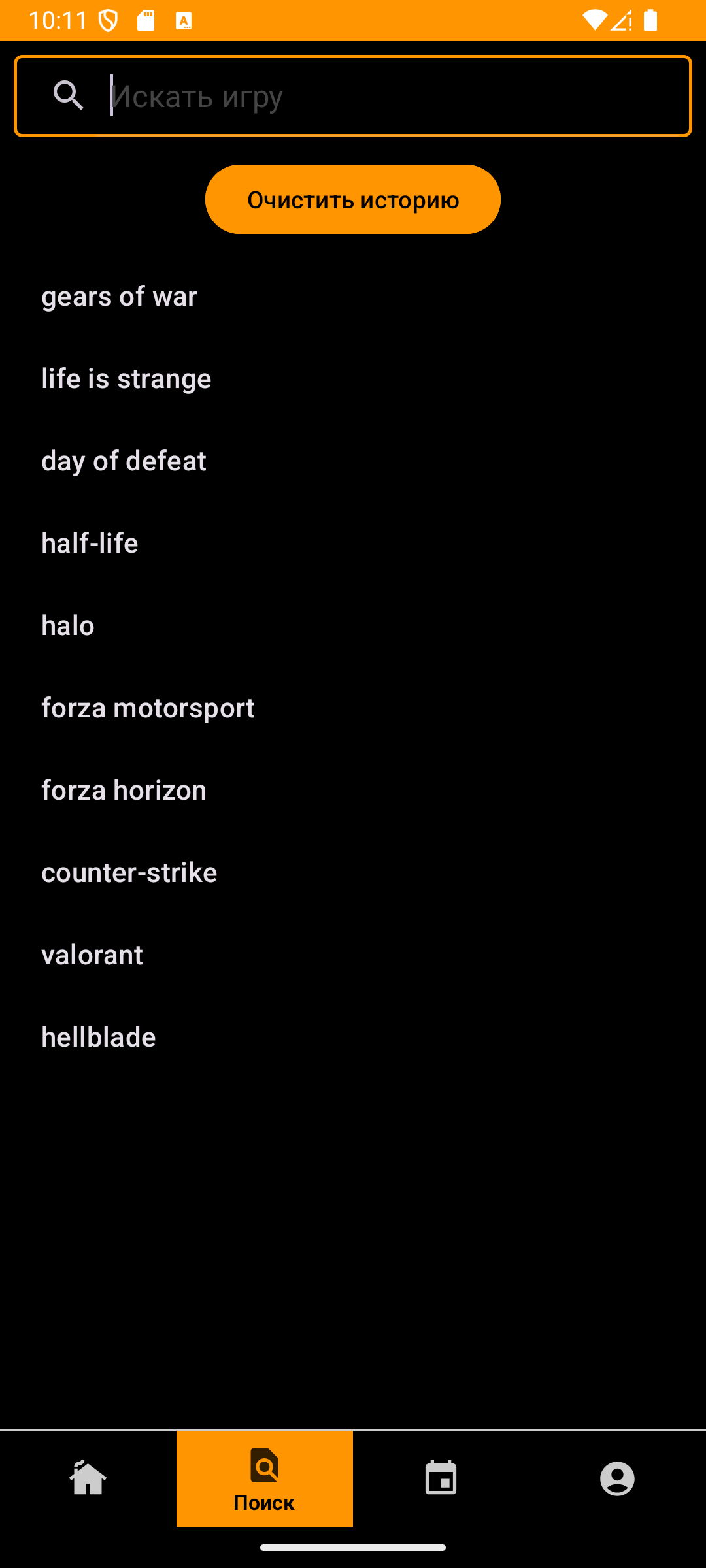


Рисунок 4 – Ограничение по количеству запросов

1. Добавление запроса в историю: при выборе элемента из списка истории, при автоматической отправке запроса или при нажатии кнопки

«Поиск» вызывается метод, который добавляет новый запрос в историю. Если такой запрос уже есть, он удаляется и вновь добавляется в начало списка – таким образом, новые запросы отображаются вверху (листинг 5, рис. 5).

*Листинг 5 – Методы загрузки и сохранения истории*

|  |
| --- |
| private fun loadHistory(): List<String> {  val prefs = requireContext().getSharedPreferences("search\_prefs", Context.MODE\_PRIVATE)  val json = prefs.getString(HISTORY\_PREF, null) ?: return emptyList()  return try {  val jsonArray = JSONArray(json)  List(jsonArray.length()) { i -> jsonArray.getString(i) }  } catch (e: Exception) {  emptyList()  }  }  private fun saveToHistory(query: String) {  val prefs = requireContext().getSharedPreferences("search\_prefs", Context.MODE\_PRIVATE)  val history = loadHistory().toMutableList()  history.remove(query)  history.add(0, query)  val limited = history.take(MAX\_HISTORY\_SIZE)  val jsonArray = JSONArray()  for (item in limited) {  jsonArray.put(item)  }  prefs.edit().putString(HISTORY\_PREF, jsonArray.toString()).apply()  } |

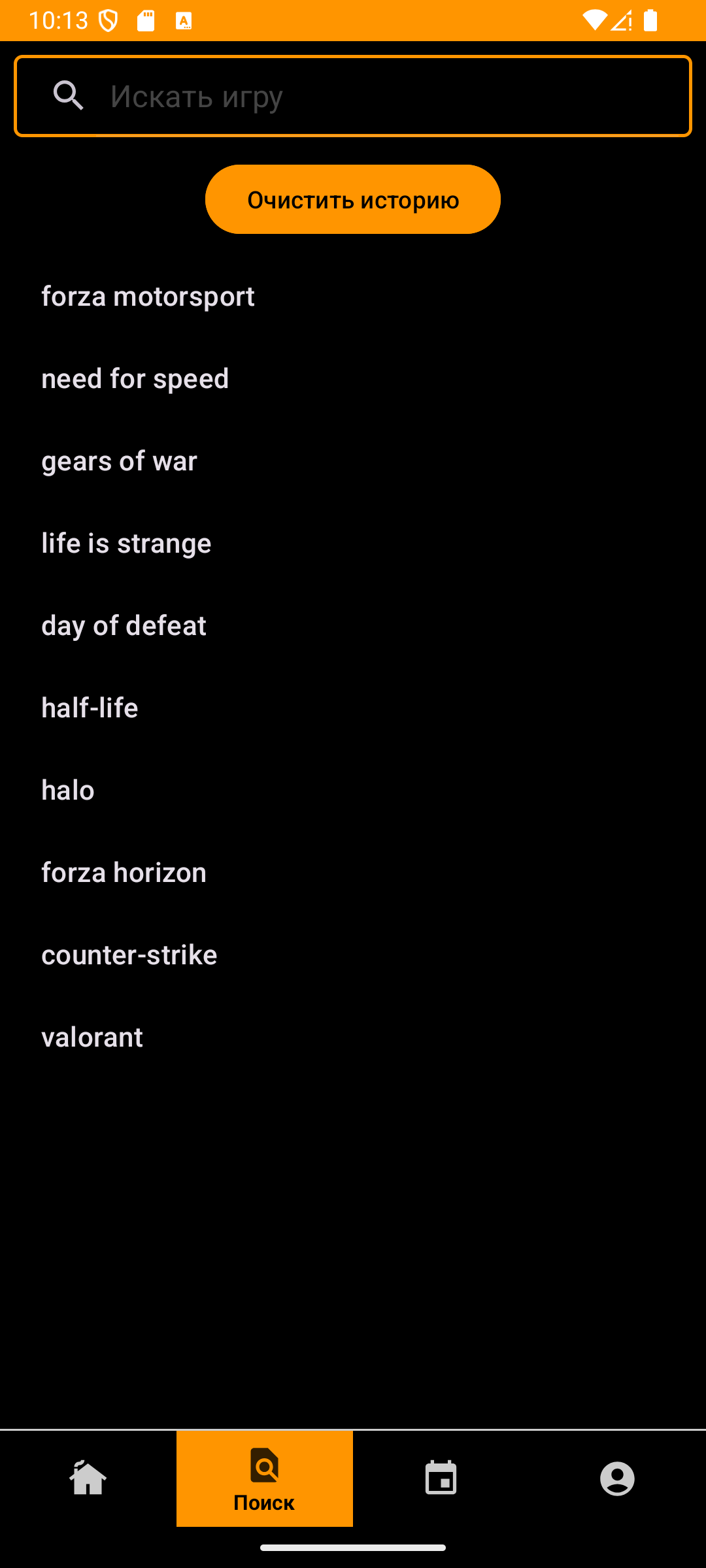
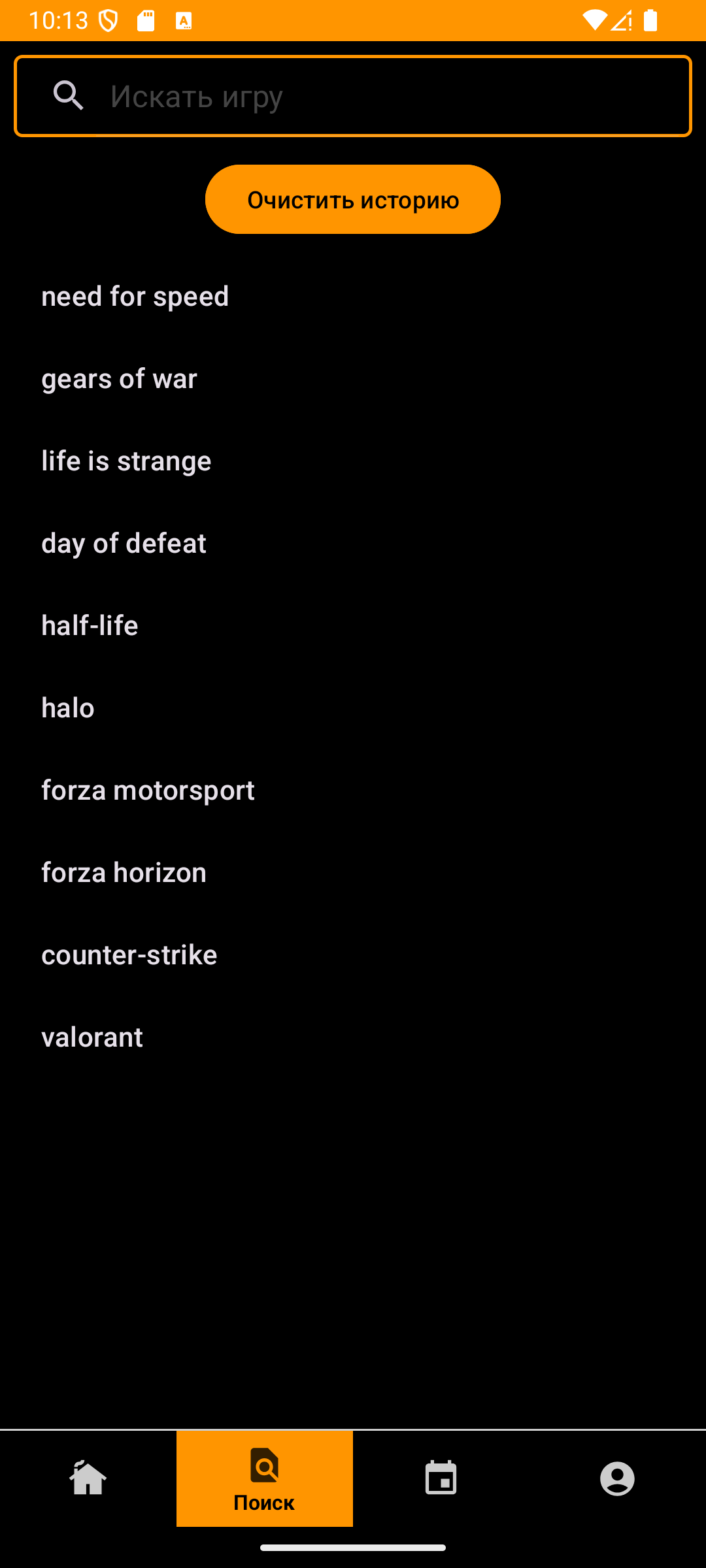


Рисунок 5 – Вызов одного из недавних запросов

1. Очистка истории: при нажатии на кнопку «Очистить историю» происходит удаление записи истории из SharedPreferences, после чего всплывающее окно закрывается (рис. 6).

*Листинг 6 – Очистка истории*

|  |
| --- |
| private fun clearHistory() {  val prefs = requireContext().getSharedPreferences("search\_prefs", Context.MODE\_PRIVATE)  prefs.edit().remove(HISTORY\_PREF).apply()  } |

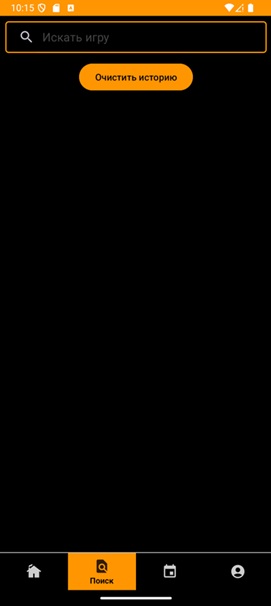
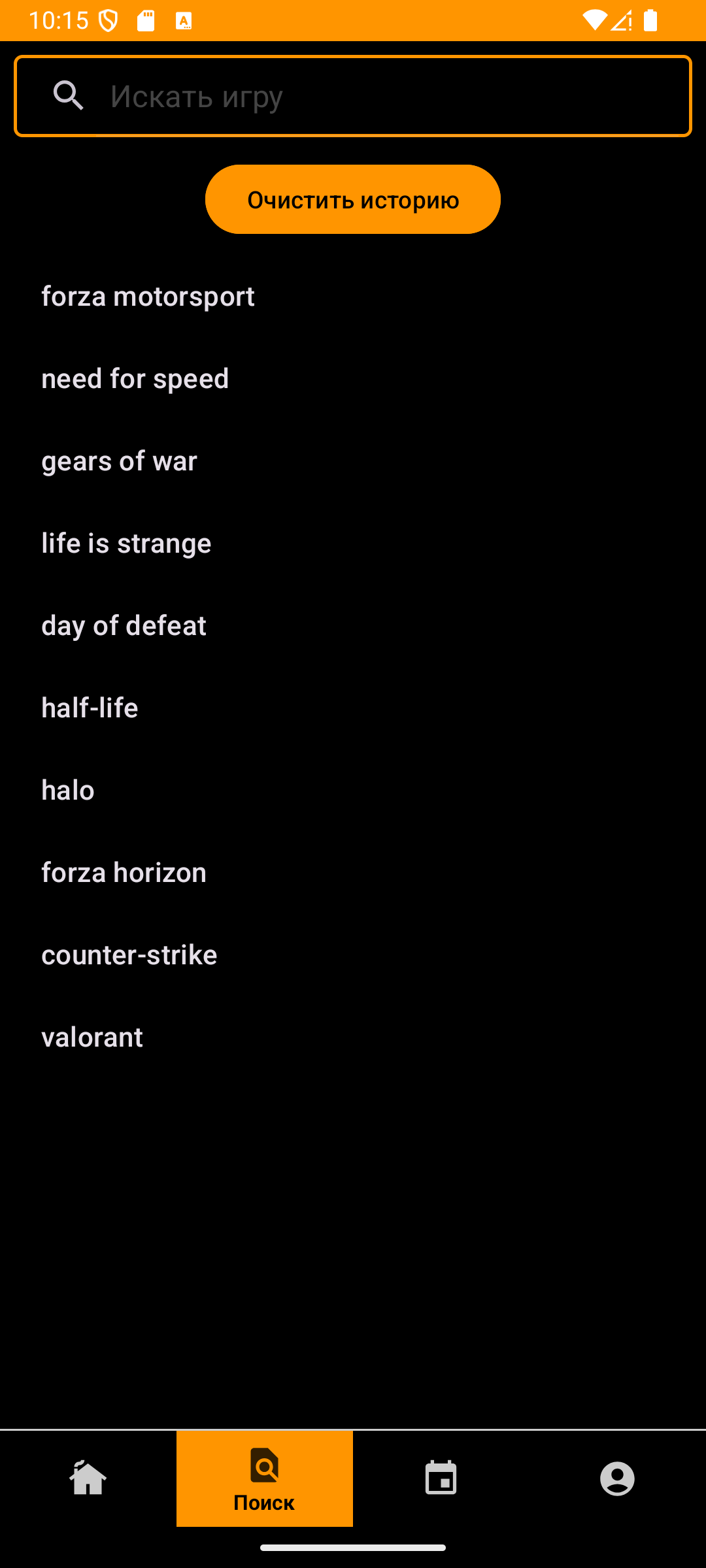
**

Рисунок 6 – Очистка истории

# Задание 3. Автоматическая отправка поискового запроса через 2 секунды бездействия

Для автоматической отправки запроса используется слушатель изменений текста (OnQueryTextListener) для поля ввода. При каждом изменении текста происходит сброс предыдущего таймера (через Handler), и, если введённый текст не пустой, запускается новый отложенный вызов (с задержкой 2000 миллисекунд). Если пользователь не вводит новые символы в течение 2 секунд, вызывается метод отправки запроса, который также добавляет запрос в историю (листинг 7, рис. 7).

*Листинг 7 – Автоматическая отправка запроса*

|  |
| --- |
| binding.searchView.setOnQueryTextListener(object : androidx.appcompat.widget.SearchView.OnQueryTextListener {  override fun onQueryTextSubmit(query: String?): Boolean {  searchRunnable?.let { handler.removeCallbacks(it) }  query?.let {  saveToHistory(it)  fetchGames(it)  hideHistory()  binding.searchView.clearFocus()  binding.searchView.postDelayed({  hideKeyboard()  }, 100)  }  return true  }  override fun onQueryTextChange(newText: String?): Boolean {  searchRunnable?.let { handler.removeCallbacks(it) }  if (newText.isNullOrEmpty()) {  if (loadHistory().isNotEmpty()) {  showHistory()  }  binding.searchView.clearFocus()  binding.searchView.postDelayed({  hideKeyboard()  }, 100)  return false  }  searchRunnable = Runnable {  saveToHistory(newText)  fetchGames(newText)  hideHistory()  }  handler.postDelayed(searchRunnable!!, SEARCH\_DELAY)  return false  }  }) |

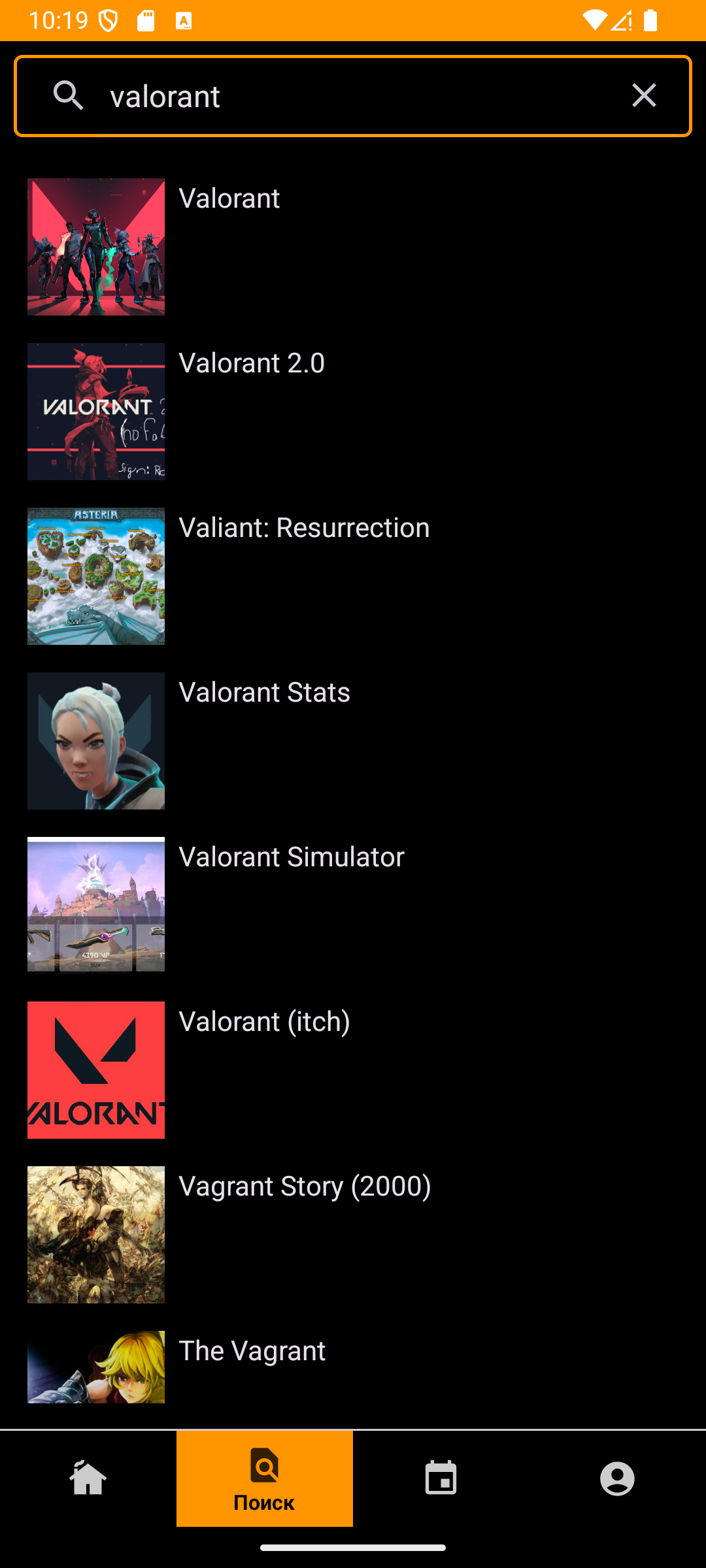
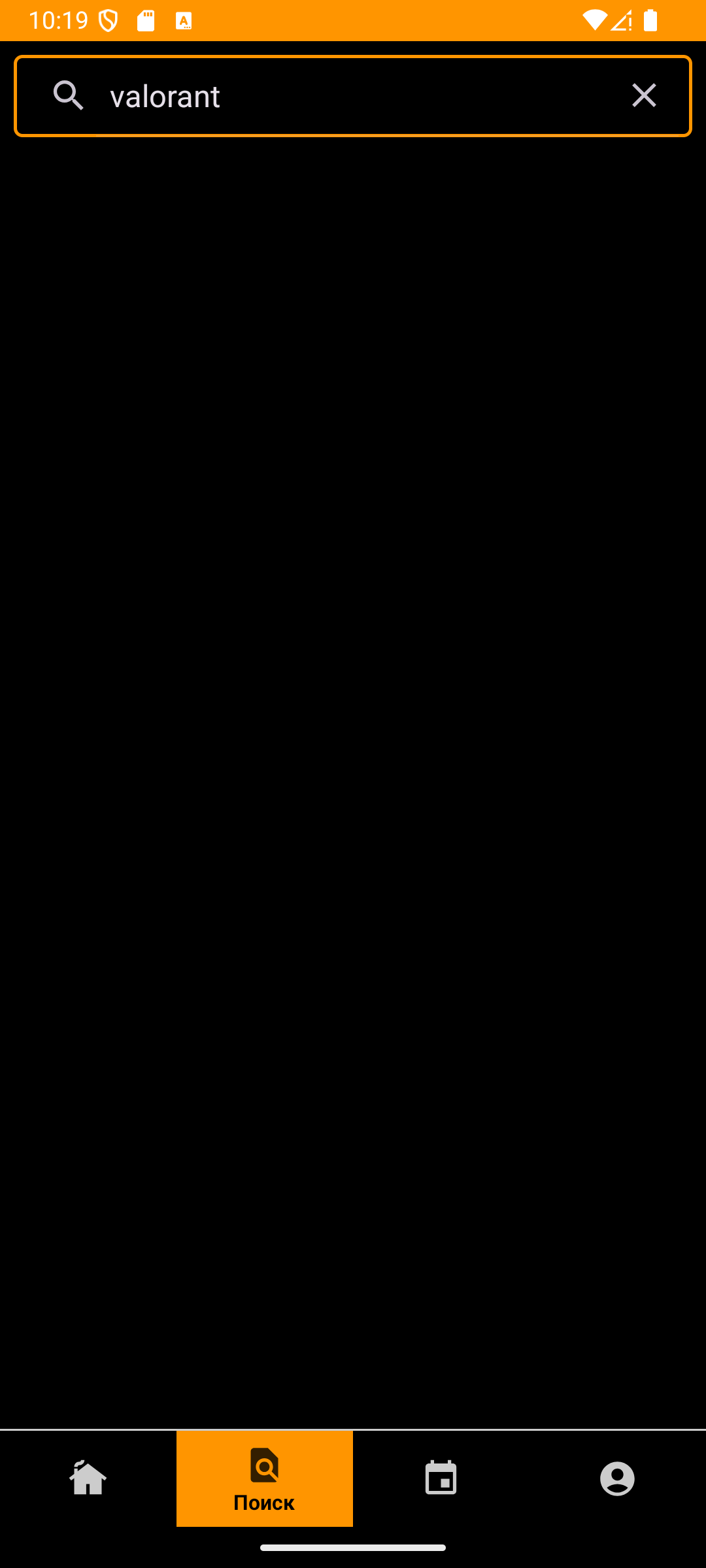


Рисунок 7 – Автоотправка запроса

# Задание 4. Отображение ProgressBar при выполнении поискового запроса

Для отображения процесса загрузки используется ProgressBar. Когда начинается запрос, вызывается функция showLoadingState(), которая:

* Скрывает RecyclerView, содержащий данные;
* Скрывает макет-заглушку placeholderLayout, если он был отображён;
* Показывает ProgressBar, сигнализируя о процессе загрузки.

*Листинг 8 – Отображение ProgressBar*

|  |
| --- |
| private fun showLoadingState() {  binding.recyclerView.visibility = View.GONE  binding.placeholderLayout.visibility = View.GONE  binding.progressBar.visibility = View.VISIBLE  } |

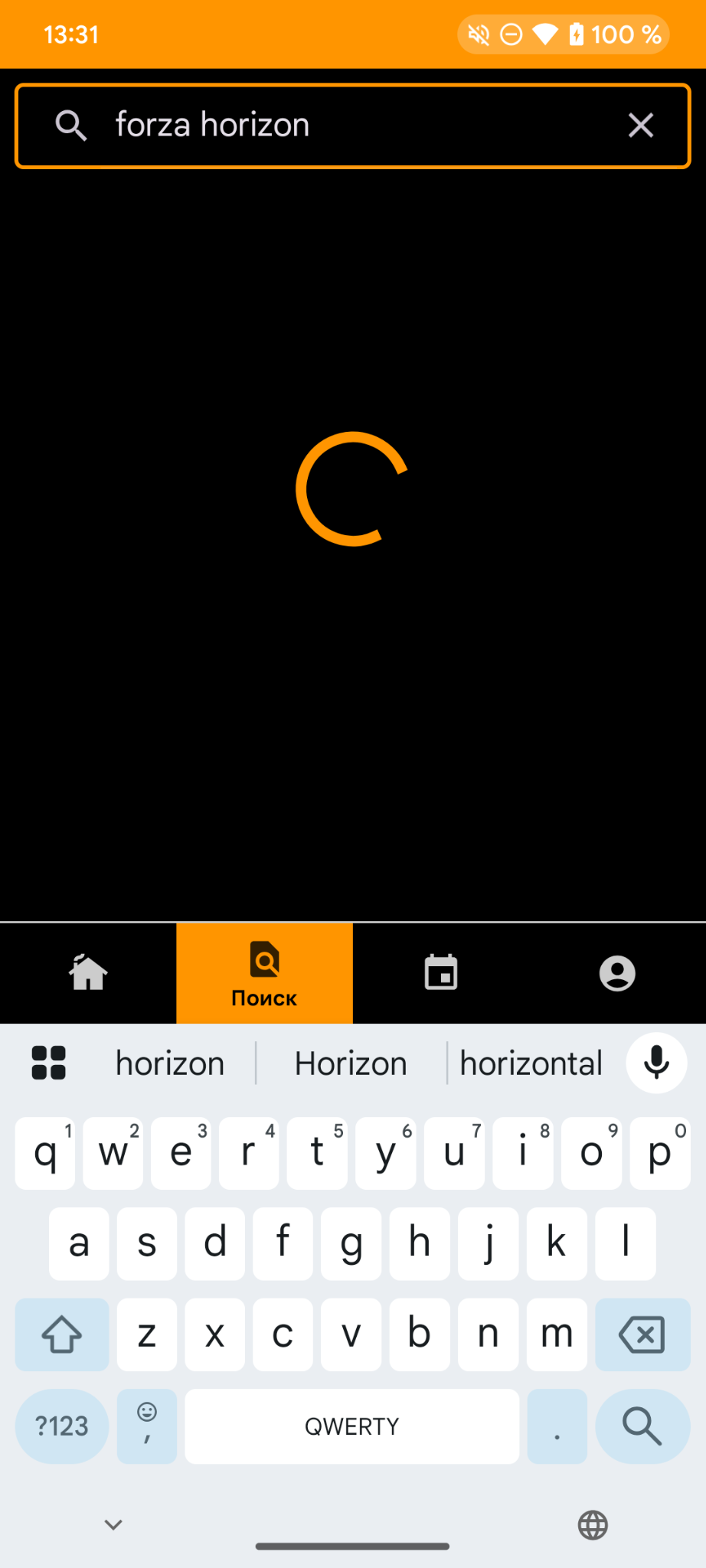
**

Рисунок 4 – Отображение ProgressBar при выполнении запроса

# Вывод

В ходе выполнения практической работы были успешно реализованы все поставленные задачи, направленные на улучшение функциональности и удобства использования мобильного приложения. Реализован переключатель тёмной темы, позволяющий пользователю адаптировать внешний вид интерфейса в соответствии со своими предпочтениями. Переключение темы происходит корректно и сохраняется между сессиями. Внедрена система хранения истории поиска с использованием SharedPreferences. При нажатии на строку поиска пользователю отображается список из последних 10 запросов (если история не пуста). История формируется в порядке добавления, при этом новые запросы отображаются первыми. При выборе элемента из результатов поиска он автоматически добавляется в историю. Также реализована функция полной очистки истории одним нажатием. Добавлена возможность автоматической отправки поискового запроса через 2 секунды бездействия пользователя после ввода. Реализовано отображение ProgressBar в момент выполнения поискового запроса. Пользователь визуально информируется о процессе загрузки, что делает интерфейс более понятным и отзывчивым.

# Список источников

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.