Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 19Б.10-мм

Власов Илья Максимович

Разработка Android приложения для поиска игроков в настольные игры

Отчёт по учебной практике

Научный руководитель:
доц. кафедры СП, к. т. н. Т. А. Брыксин

Консультант:
программист-исследователь JetBrains Е. С. Спирин

Санкт-Петербург
2021

Оглавление

1 Введение

В мире существует большое количество настольных игр разных жанров и с каждым годом их популярность растёт. Однако не всем нравятся все жанры. Поэтому у игроков возникает проблема с поиском других игроков поблизости с похожими интересами.

Предположим, что мы купили игру, но нам не с кем поиграть. Для того, чтобы исправить эту проблему мы идём в интернет в надежде найти партнёров. При этом мы хотим найти игроков как можно ближе и с похожими интересами.

Однако сделать этого не можем, так как в интернете нет удобного средства для поиска игроков в настольные игры: социальные сети и приложения для знакомств не подходят, а готовых приложений нет.

Поэтому для решения этой проблемы было решено написать клиент-серверное приложение для поиска игроков в настольные игры.

С помощью приложения пользователь сможет:

- Авторизоваться с помощью ВКонтакте;
- Персонализировать свой профиль, указав местоположение и предпочтения;
- Найти игроков поблизости;
- Добавить других пользователей в список друзей.

В рамках данного проекта описывается реализация клиента данного приложения под Android.

2 Цель и задачи

Цель

Целью работы является создание клиентского приложения под Android для поиска игроков в настольные игры.

Задачи

Для выполнения цели были поставлены следующие задачи:

1. Изучить аналоги приложения;
2. Разработать архитектуру клиентского приложения с учётом API сервера;
3. Реализовать приложение:
   (a) Реализовать систему авторизации пользователя;
   (b) Реализовать поиск других игроков;
   (c) Реализовать систему друзей.
4. Провести тестирование на реальных пользователях.

3 Обзор

3.1 Существующие решения

При анализе рынка было найдено лишь одно приложение похожее по функциональности: *GameFindr*[1]. Однако данное приложение не обновляется с 2015 года и похоже заброшено. Сервера приложения отключены, официальный сайт не работает. Поэтому возможности проанализировать приложение нет.

Функционал социальных сетей не предоставляет удобный способ поиска игроков.

С одной стороны, можно создать группу по интересам, однако в группе будут находится люди из разных городов.

С другой стороны, можно создать группу определённого города, но тогда людям будет трудно искать других игроков со схожими интересами.

Наконец, можно создать группу одновременно для определённого города и для определённых интересов, но, во-первых, таких групп понадобится очень много, во-вторых, если пользователь захочет поменять город или у него поменялись интересы, то придётся искать другую группу, что не удобно.

Приложения для знакомств тоже не помогут, так как пользователю чаще всего будут показываться люди, которые хотят просто познакомиться, а не поиграть в настольные игры.

Таким образом, мы не смогли найти приложение под поставленные требования.

3.2 Background

Было решено написать приложение под Android, так как сейчас Android является самой распространённой ОС для мобильных устройств с долей рынка в 72,48%[2].

[1] https://apkpure.com/ru/gamefindr/com.ht.gamefindr

[2] https://gs.statcounter.com/os-market-share/mobile/worldwide

В качестве языка был выбран Kotlin, так как данный язык сегодня является основным при разработке приложений под Android.

При разработке были использованы следующие библиотеки:

- **VK SDK для Android** [9].

  Используется для авторизации пользователя, потому что в этой библиотеке уже реализован весь процесс авторизации.

- **Retrofit2** [8].

  Используется для общения с сервером, потому что это типобезопасный HTTP-клиент для Android, который предоставляет готовый удобный интерфейс к API запросам.

- **Paging Library 3** [7].

  Используется для создания динамических списков, потому что именно эту библиотеку Google рекомендует[3]использовать в Android приложениях.

[3] https://developer.android.com/topic/libraries/architecture

4 Архитектура

Так как приложение подразумевает клиент-серверное взаимодействие, то было важно разграничить взаимодействие с пользователем и взаимодействие с сервером, поэтому было решено использовать MVVM (Model-View-ViewModel) архитектуру. Она позволяет разделить приложение на 3 части (см. рис. 1):

- **Model** — основная логика приложения;
- **View** — пользовательский интерфейс;
- **ViewModel** — модель представления данных, которая, с одной стороны, является абстракцией интерфейса, а с другой – обёртка данных из модели.

View и ViewModel связанны друг с другом с помощью процесса «связывания данных» (data binding). Благодаря этому, любые изменения произошедшие во ViewModel сразу же отразятся во View, и наоборот.

При этом View знает только о существовании ViewModel, а ViewModel — только о Model. Таким образом, имеется возможность тестирования ViewModel и Model независимо от Android окружения.

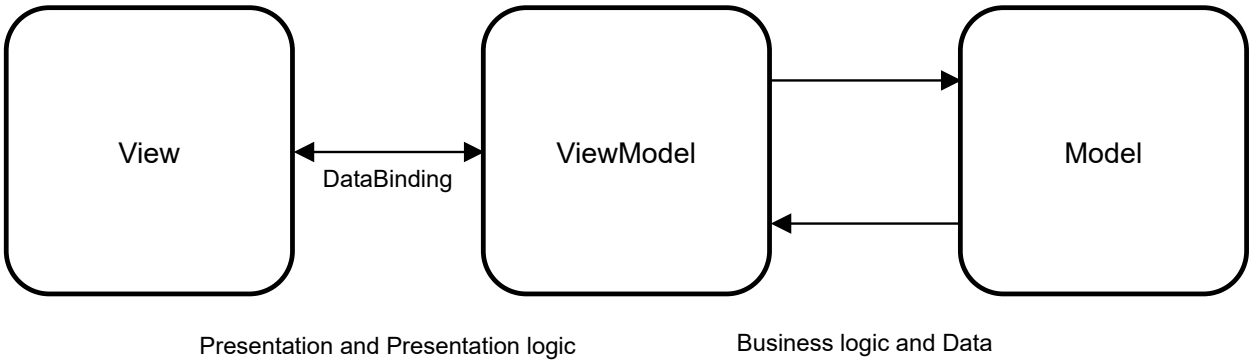

Рис. 1: Диаграмма MVVM архитектуры

4.1 Взаимодействие с серверной частью

Если ViewModel хочет обратиться к серверу, то он вызывает нужный метод из *ServerRepository*, передавая какие-то данные (например, ID

пользователя). Далее, *ServerRepository* с помощью Retrofit2 обращается к серверу, а полученный ответ он возвращает ViewModel уже обёрнутый в *LiveData* (см. рис. 2).

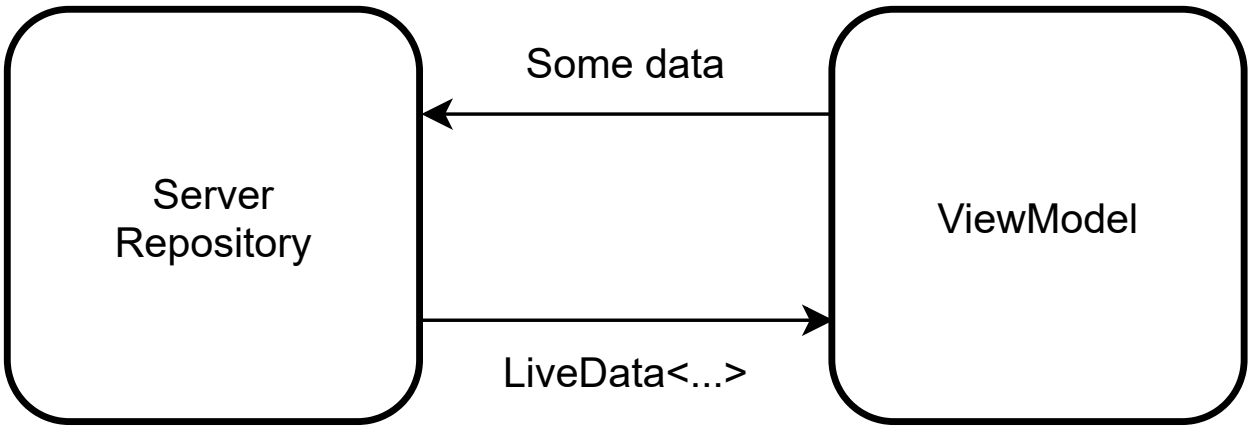

Рис. 2: Взаимодействие *ServerRepository* с ViewModel

LiveData [4] — реализация паттерна «Наблюдатель». Так как приложение не знает, когда с сервера придёт ответ, то оно просто будет наблюдать за *LiveData* и как только её содержимое изменится, выполнит необходимые действия.

Так как во время отправки запроса или получения ответа, могут возникнуть какие-то ошибки (например, ошибка соединения или ошибка десериализации), приложению нужно их как-то обрабатывать. Для этого все данные, которые возвращает *ServerRepository* наследуются от единого класса *ServerRepositoryResponse*, в частности от него наследуется класс *ServerError*, который хранит в себе тип и описание произошедшей ошибки.

Далее, ViewModel уже с помощью класса *ErrorHandler* узнает, является ли ответ сервера ошибкой или нет (см. рис. 3). Если да, то ViewModel с помощью того же *ErrorHandler* обрабатывает ошибку, получая *Event*, который далее передаётся во *View* для дальнейшей обработки.

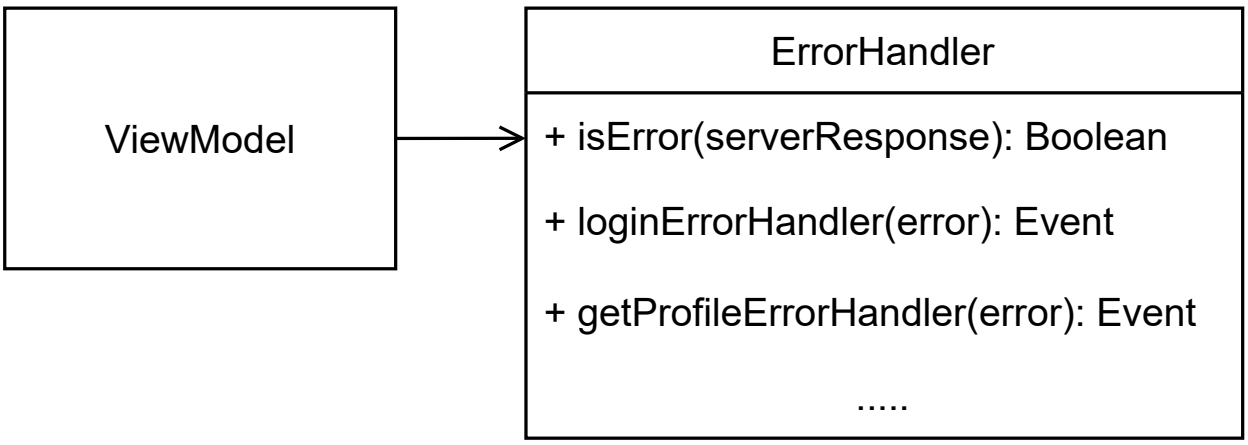

Рис. 3: Структура ErrorHandler

5 Реализация

5.1 Авторизация и персонализация

После того, как пользователь прошёл процедуру авторизации, приложение получает *access token* от ВКонтакте, который далее передаётся на сервер. Сервер же возвращает, или *server token* (это означает, что пользователь уже присутствует в базе данных), или сообщение о том, что такого пользователя нет.

Если приложение получило *server token*, то пользователю показывается его профиль, иначе — страница с персонализацией, где пользователя просят указать его местоположение, предпочтения и просят написать что-нибудь о себе.

Пользователю не требуется при каждом запуске приложения проходить процедуру авторизации, так как приложение запоминает информацию о том, вошёл ли пользователь в приложение или нет.

С помощью *EncryptedSharedPreferences* [2] токен в зашифрованном виде хранится на устройстве пользователя. *Server token* сгорает только в том случае, если пользователь выходит из приложения, нажимая на кнопку «Выйти». В таком случае, приложение попросит пользователя заново войти с помощью ВКонтакте, чтобы получить новый *server token*.

5.2 Поиск игроков

Так как вокруг пользователя может находится огромное количество других игроков, то было бы плохим решением сразу получать с сервера всех игроков, так как загрузка такого объема данных может занять много времени для пользователей имеющих плохое интернет-соединение.

Поэтому для решения данной проблемы приложение получает данные кусками по 50 человек и динамически запрашивает и показывает новые части по мере необходимости.

50 человек хватает для того, чтобы экран поиска был полностью заполнен. При этом у пользователя в запасе находится ещё несколько десятков человек, поэтому приложению не приходится непрерывно обращаться к серверу.

Для реализации динамических списков я использую библиотеку Paging Library 3 [7]. Для того, чтобы реализовать динамические списки, мне потребовалось 4 компонента библиотеки (см. рис. 4):

1. *PagingData* — контейнер для данных с разбивкой на страницы. Каждому обновлению данных соответствует отдельный *PagingData*;

2. *PagingSource* — базовый класс для загрузки снимков данных в поток *PagingData*;

3. *Pager.liveData* — строит *LiveData<PagingData>* на основе PagingConfig и *PagingSource*;

4. *PagingDataAdapter* — *RecyclerView.Adapter*, который представляет *PagingData* в *RecyclerView*. *PagingDataAdapter* прослушивает внутренние события *PagingData* по мере загрузки страниц и использует *DiffUtil* [1] в фоновом потоке для выявления обновлений по мере получения обновленного содержимого в форме новых объектов *PagingData*.

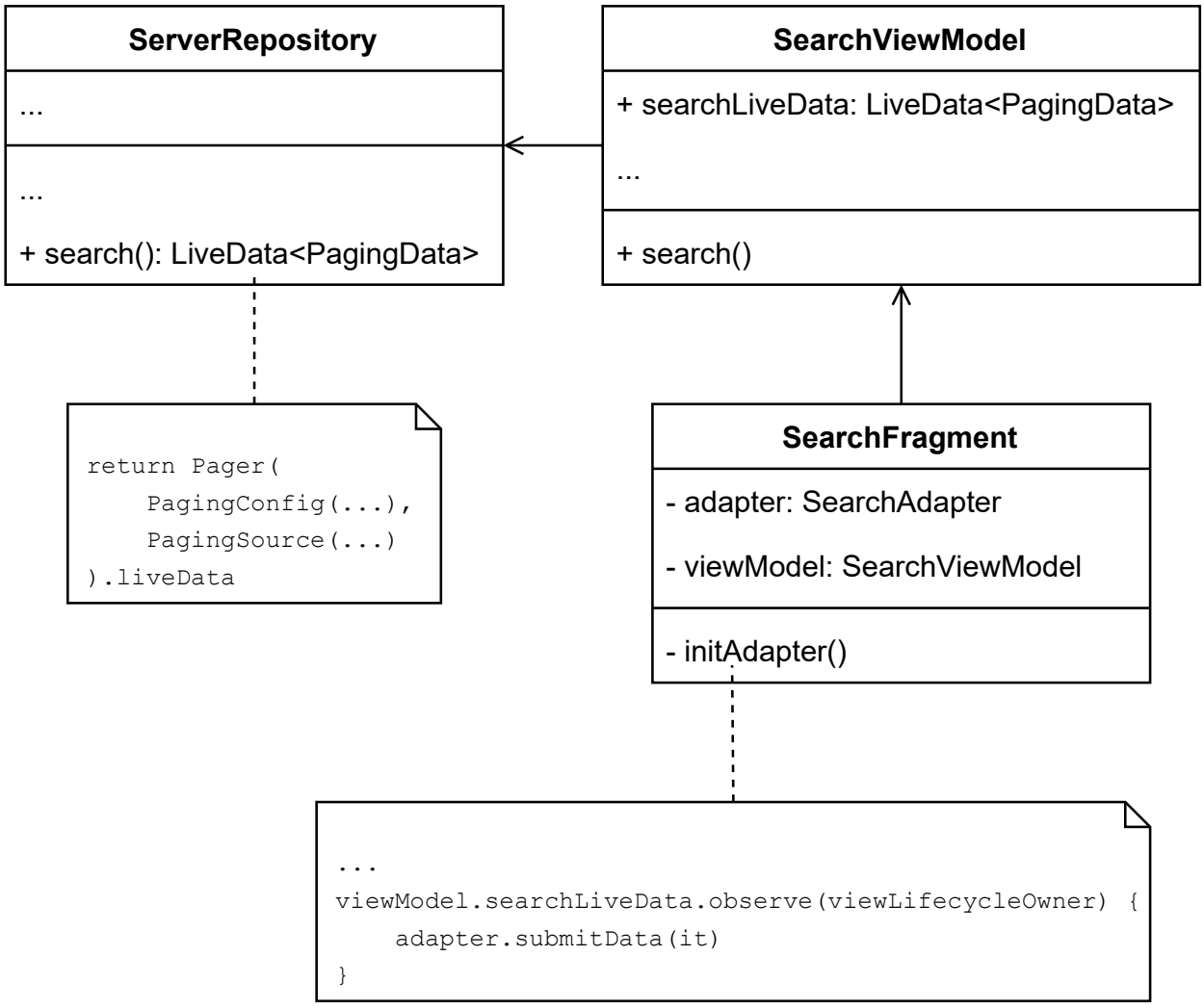

Рис. 4: Интеграция Paging Library в существующую архитектуру

5.3 Система друзей

С сервера, помимо данных профиля другого игрока, приложение получает код *friendStatus*. Данный код характеризует отношения между пользователями (см. таб. 1).

friendStatus	**Значение**
0	Нет отношений
1	Запрос получен от пользователя
2	Запрос отправлен пользователю
3	Друзья

Таблица 1: Значения *friendStatus*

Так как вид и поведение профиля игрока зависит от *friendStatus*, то при реализации профиля был использован паттерн «Состояние» (см. рис. 5).

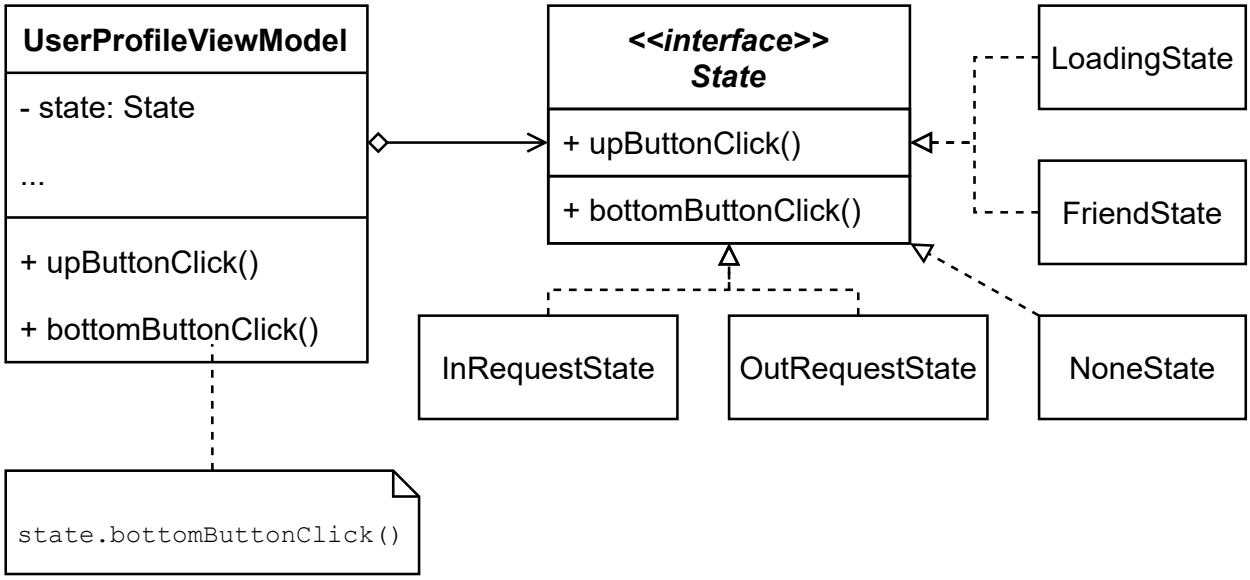

Рис. 5: Диаграмма паттерна «Состояние»

После того как два игрока подружились, они могу связаться друг с другом с помощью кнопки ”Написать”, по нажатию на которую открывается соответствующая страница с профилем ВКонтакте. Дальнейшее общение игроков происходит с помощью чата ВКонтакте.

Списки «Друзья», «Отправленные запросы», «Полученные запросы» и «Отклоненные запросы» были реализованы аналогично списку «Поиск», который был подробно рассмотрен в секции 5.2.

6 Тестирование

Для тестирования были использованы:

- **JUnit** [3]. Для создания Unit тестов.
- **MockWebServer** [6]. Для тестирования API запросов.

На данный момент с помощью Unit-тестов протестировано взаимодействие клиента и сервера. В будущем планируется полное покрытие проекта тестами.

Было проведено ручное тестирование всех сценариев использования приложения (авторизация, регистрация, редактирования каждого пункта профиля, принудительное обновление профиля пользователя или другого игрока, запрос списка ближайших игроков, друзей, отправленных запросов, полученных запросов и отклонённых запросов, отправка/отзыв/принятие/отклонение запроса в друзья). Также было протестировано поведение приложения в нештатных ситуациях (отсутствие интернет-соединения, отсутствие ответа с сервера).

Помимо этого было проведено тестирование на реальных пользователях. Было задействовано около 20 человек.

7 Заключение

Таким образом, было написано и протестировано на реальных пользователях приложение под Android [5] для поиска игроков в настольные игры.

Скриншоты приложения можно найти в секции 8.

7.1 Результаты

При работе были выполнены следующие задачи:

1. Изучены аналоги приложения;
2. Разработана архитектура клиентского приложения с учётом API сервера;
3. Реализовано приложение:
   (a) Реализована система авторизации пользователя;
   (b) Реализован поиск других игроков;
   (c) Реализована система друзей.
4. Проведено тестирование на реальных пользователях.

7.2 Задачи на будущее

Возможное развитие:

- Добавить возможность сортировки пользователей;
- Обновить дизайн приложения;
- Написать собственный чат;
- Добавить другие способы авторизации.

8 Приложение

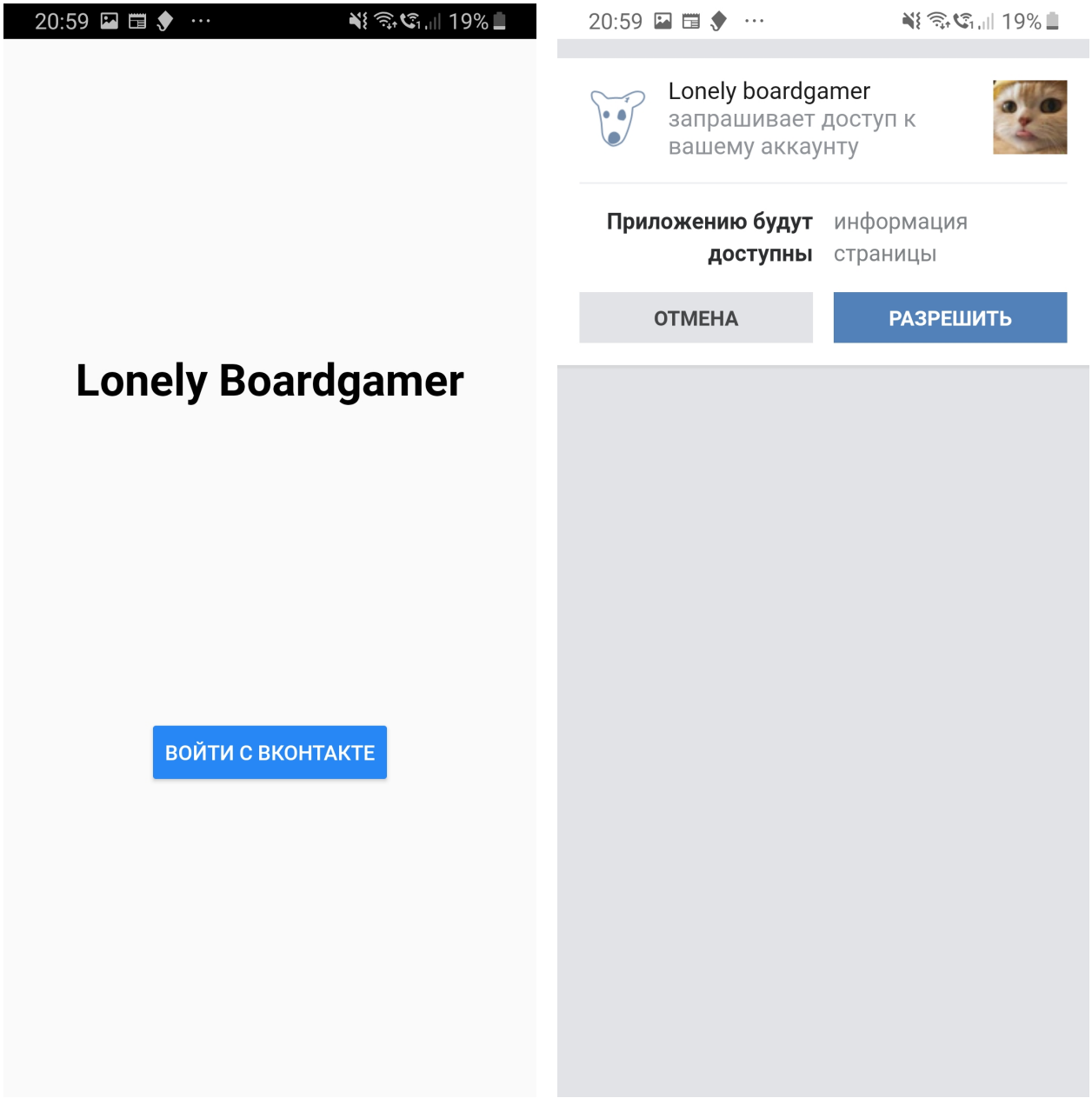

Рис. 6: Экран авторизации

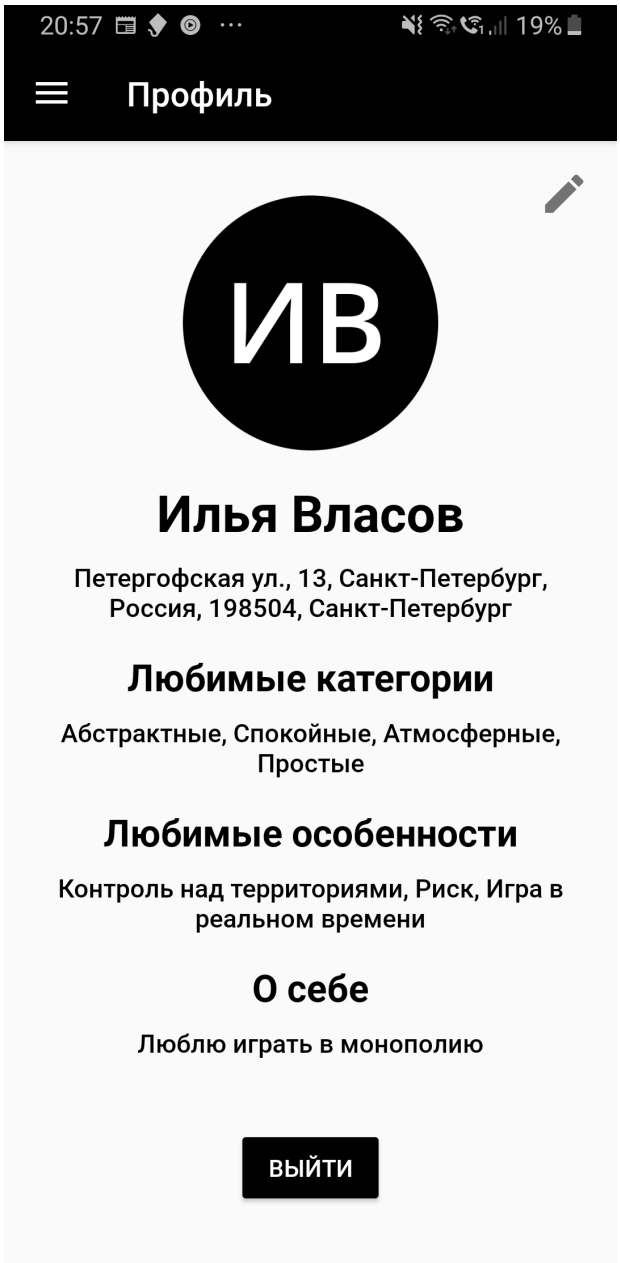

Рис. 7: Профиль пользователя

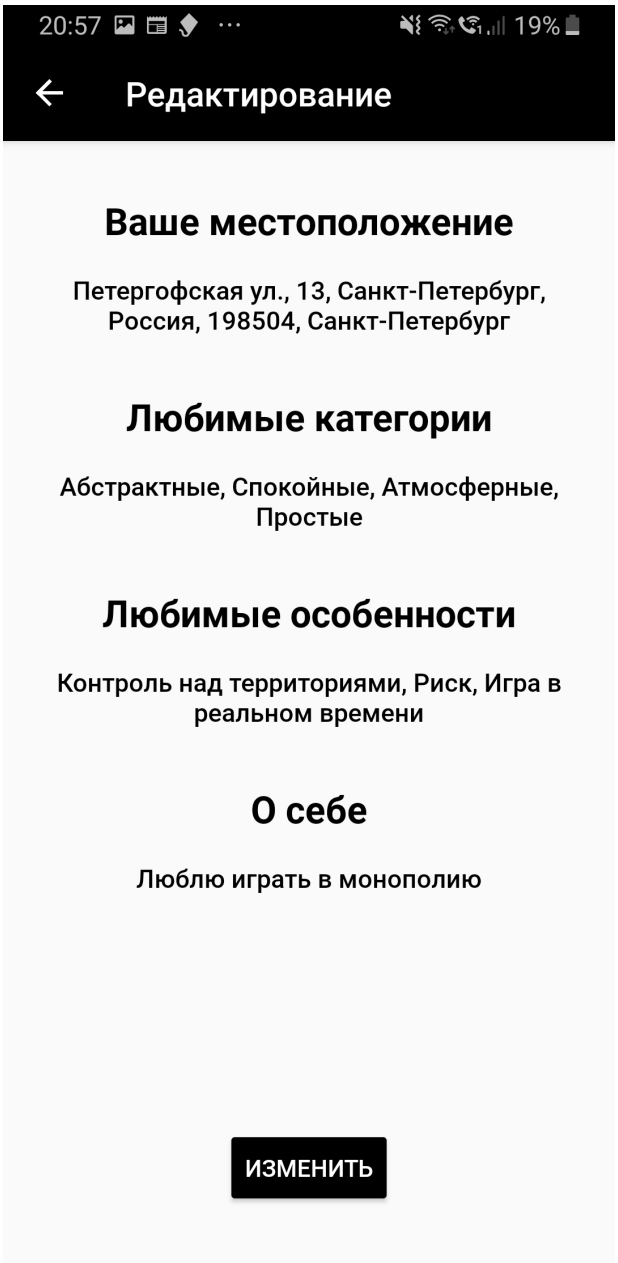

Рис. 8: Редактирование профиля

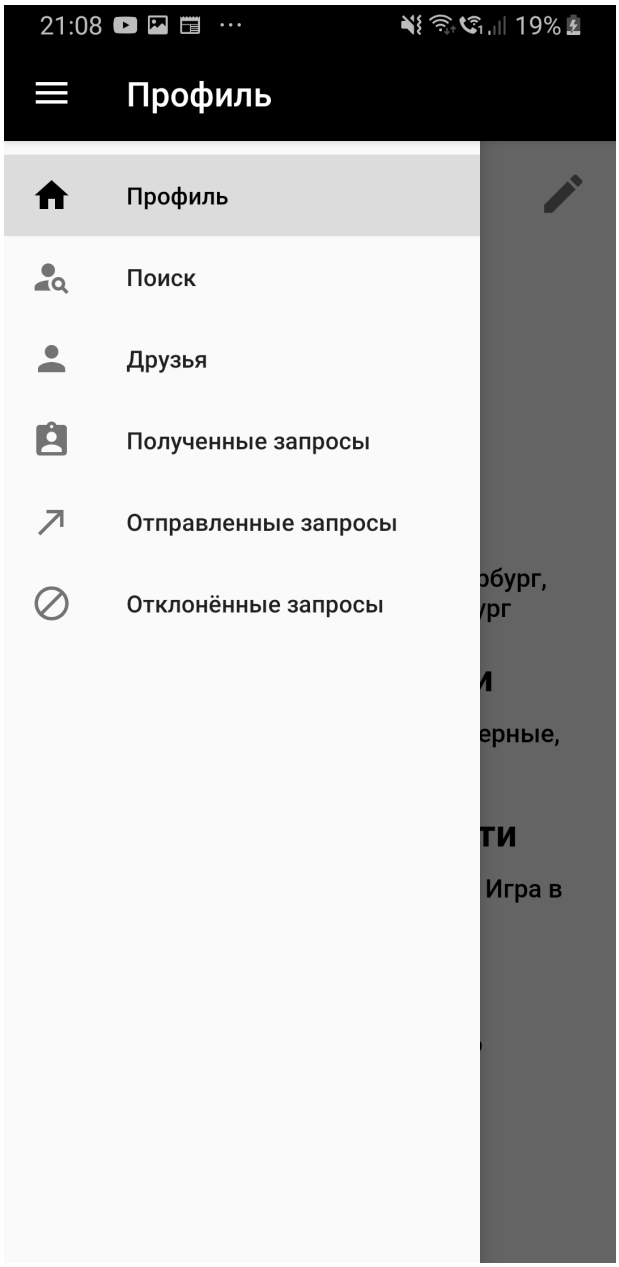
21:08
19%
Профиль
Профиль
Поиск
Друзья
Полученные запросы
Отправленные запросы
Отклонённые запросы

Рис. 9: Меню

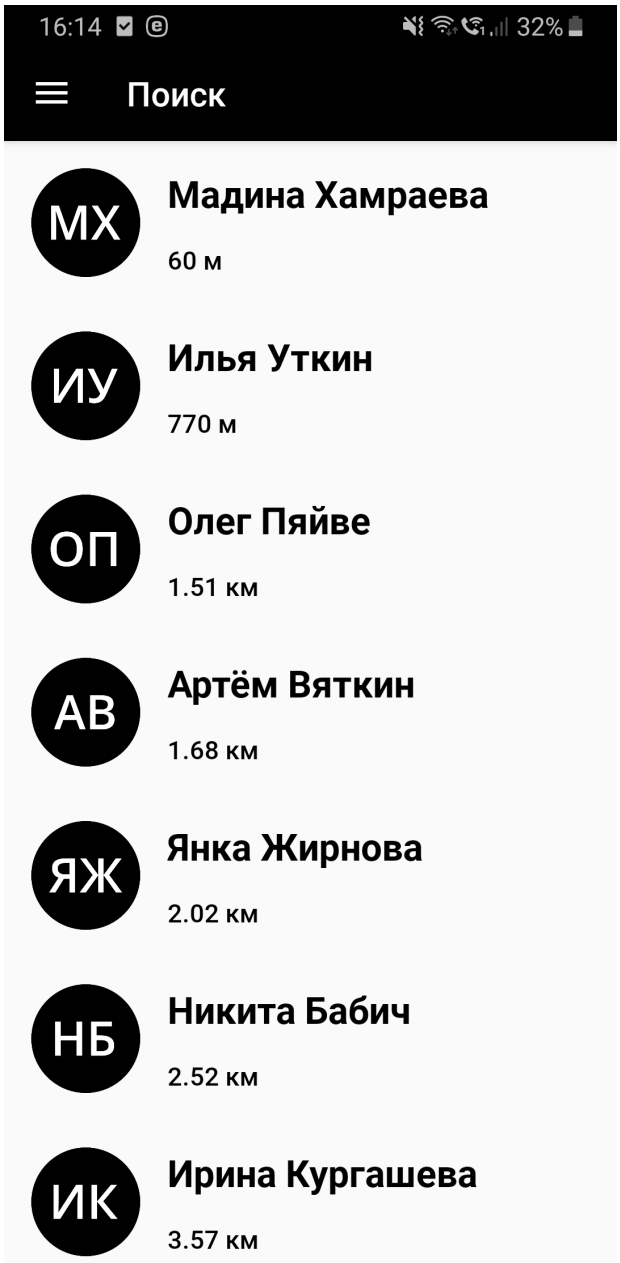

Рис. 10: Страница поиска

Рис. 11: Профиль игрока

Рис. 12: Список друзей

Рис. 13: Профиль друга

Рис. 14: Полученные запросы

Рис. 15: Профиль игрока

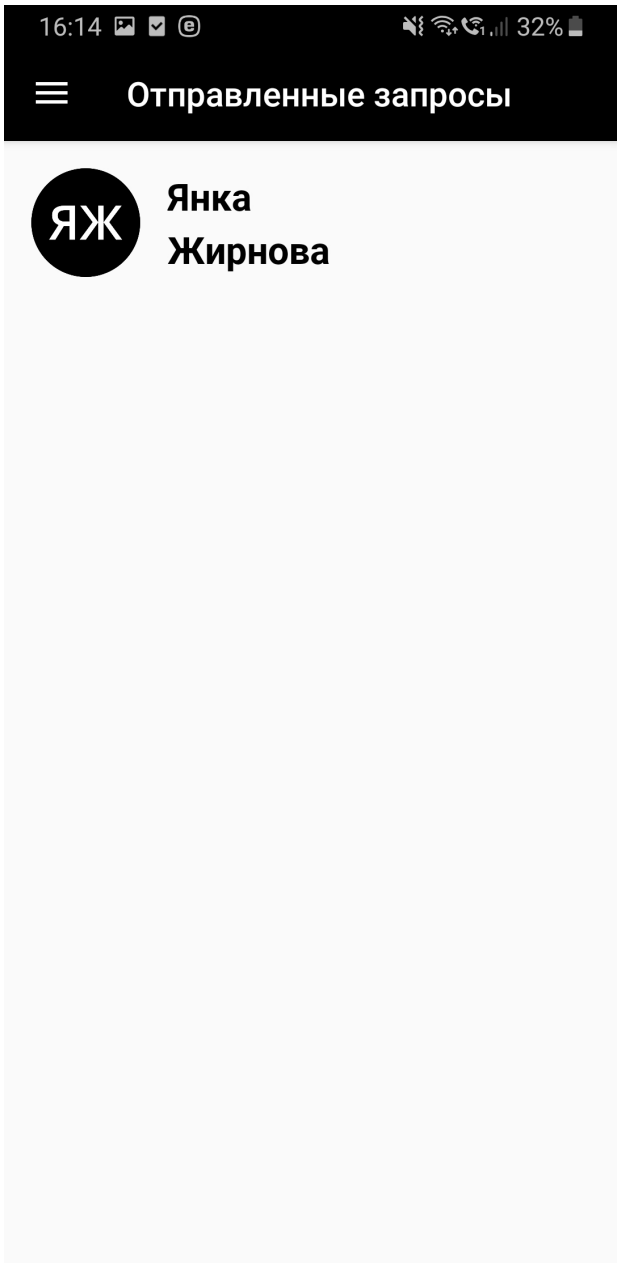

Рис. 16: Отправленные запросы

Рис. 17: Профиль игрока

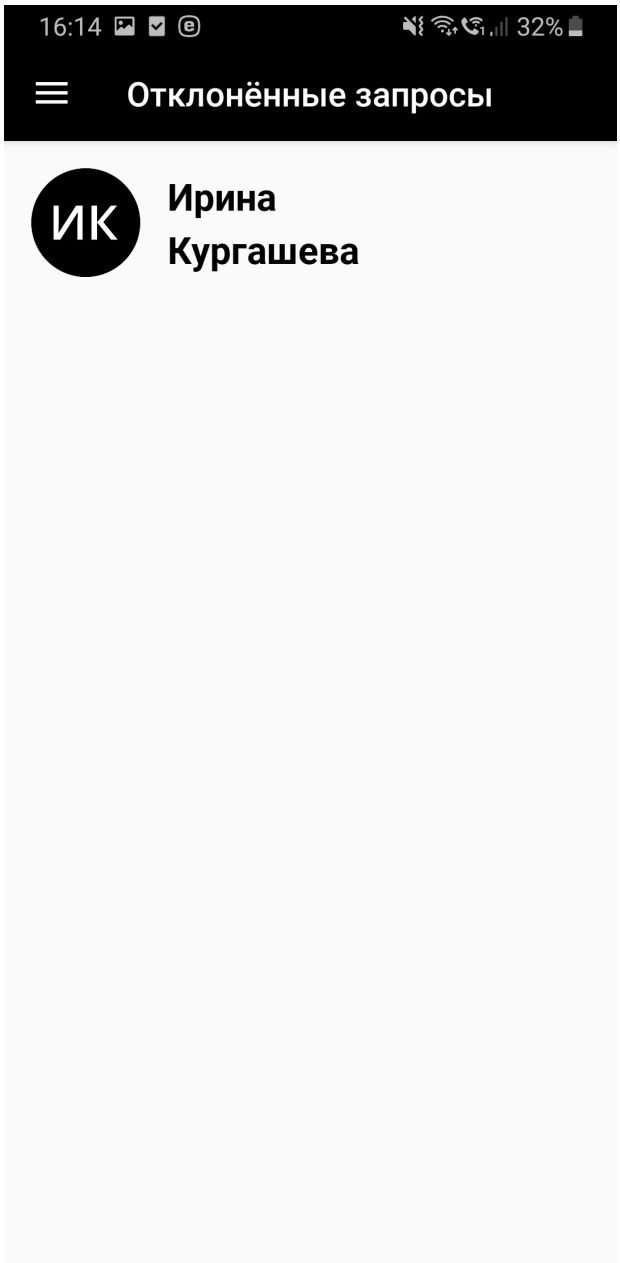

Рис. 18: Отклонённые запросы

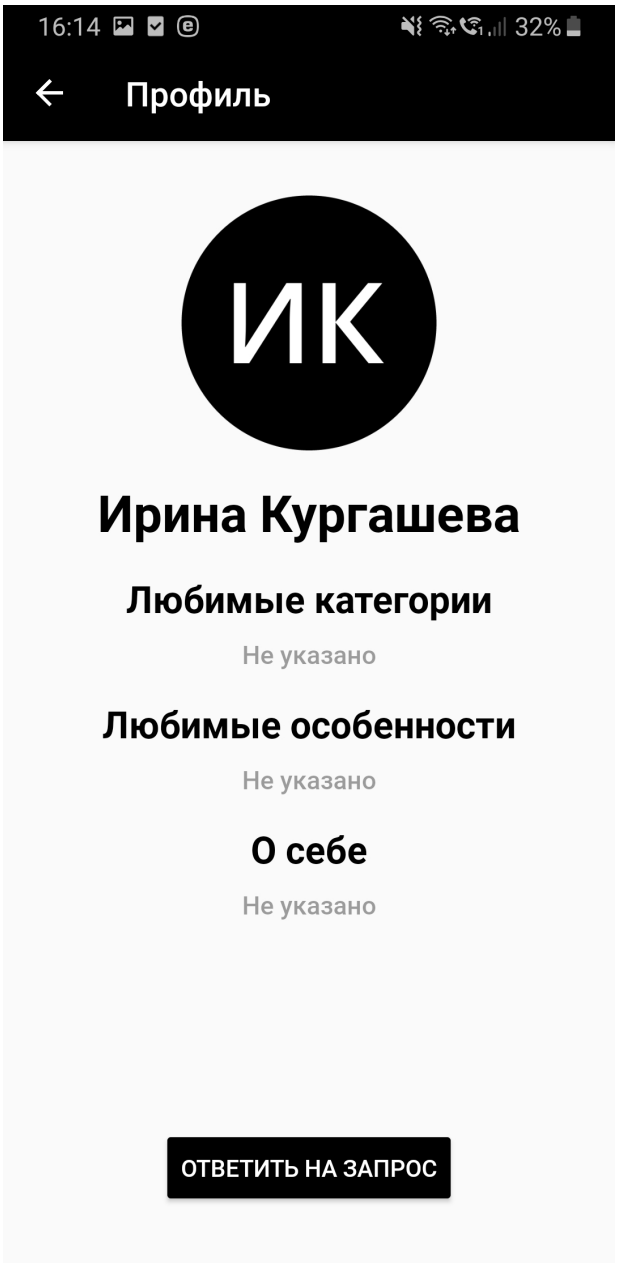

Рис. 19: Профиль игрока

Список литературы

[1] DiffUtil. — Access mode: https://developer.android.com/reference/androidx/recyclerview/widget/DiffUtil (online; accessed: 2020-12-25).

[2] Encrypted Shared Preferences. — Access mode: https://developer.android.com/topic/security/data (online; accessed: 2021-01-04).

[3] JUnit. — Access mode: https://junit.org (online; accessed: 2021-01-05).

[4] LiveData. — Access mode: https://developer.android.com/topic/libraries/architecture/livedata (online; accessed: 2020-12-25).

[5] Lonely Boardgamer source code. — Access mode: https://github.com/GirZOn/LonelyBoardgamerAndroidApp (online; accessed: 2021-01-03).

[6] MockWebServer. — Access mode: https://github.com/square/okhttp/tree/master/mockwebserver (online; accessed: 2021-01-05).

[7] Paging Library 3. — Access mode: https://developer.android.com/topic/libraries/architecture/paging/v3-overview (online; accessed: 2020-12-25).

[8] Retrofit. — Access mode: https://square.github.io/retrofit (online; accessed: 2020-12-25).

[9] VK SDK для Android. — Access mode: https://vk.com/dev/android_sdk (online; accessed: 2020-12-25).