

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА

на тему

**Мобильное приложение для организации
записи и просмотра игрового контента с
использованием потокового видео**

Студент

 18.06.21 Маркичев Кирилл Александрович
(подпись, дата) (ФИО полностью)

Руководитель

 18.06.21 Князев В.И.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 22.06.21 Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой

(подпись) Макарычев П.П.
(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК

(подпись) Попова Н.А.
(фамилия, инициалы)

Пенза, 2021

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
П.П. Макарычев
«__» _____ 2020 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Маркичев Кирилл Александрович гр. 17ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео

Тема утверждена приказом ПГУ № _____ /о от "28" декабря 2020

3. Руководитель работы _____ к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Разработать мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео, позволяющее записывать видео игрового процесса с экрана мобильного устройства, просматривать видео других пользователей в режиме реального времени, а также содержащее функционал подписок на других пользователей

2) Среда выполнения: Операционная система Android 7.0 и выше для клиентской части, операционная система Windows 10 для серверной части

Языковые средства разработки: Kotlin, Java, Spring Boot, SQL, UML

Инструментальные средства разработки: Android Studio 4.1.1, IntelliJ IDEA Community Edition 2020.3, смартфон Honor Play, Microsoft Project, Microsoft Visio, ERwin Data Modeler

Технологии разработки: RUP, ООП, MVVM, Rest

3) Основные функции мобильного приложения:

а) запись видео с экрана устройства;

б) просмотр видео других пользователей;

- б) просмотр видео других пользователей;
 в) подписка на других пользователей;
 г) просмотр статистики профиля;
 д) организация чата для потокового видео.

5. Перечень подлежащих разработке вопросов

- 1) Анализ предметной области и постановка задач
 2) Проектирование
 3) Реализация
 4) Контроль качества программного обеспечения
 5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и постановка задачи	15 %	14.04.2021 - 22.04.2021	<i>ВМ</i>
2	Проектирование	25 %	23.04.2021 - 06.05.2021	<i>ВМ</i>
3	Реализация	40 %	07.05.2021 - 31.05.2021	<i>ВМ</i>
4	Тестирование	5 %	01.06.2021 - 03.06.2021	<i>ВМ</i>
5	Оформление пояснительной записки	15 %	04.06.2021 - 14.06.2021	<i>ВМ</i>

Дата выдачи задания "28" декабря 2020

Руководитель бакалаврской работы *ВМ* 28.12.2020 г.

(подпись, дата)

Задание к исполнению принял студент *МФ* 28.12.2020
 (подпись, дата)

Работу к защите допустить

Декан факультета

(подпись, дата)

Реферат

Пояснительная записка содержит 127 листов, 63 рисунка, 13 таблиц, 30 использованных источников и 5 приложений.

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ПОТОКОВОЕ ВИДЕО, RUP, MVVM, REST API, KOTLIN, JAVA, SPRING BOOT

Объектом разработки является мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео.

Цель выпускной квалификационной работы – разработать мобильное приложения для организации записи и просмотра игрового контента с использованием потокового видео.

В результате выполненной работы создано мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео, позволяющее записывать видео игрового процесса с экрана мобильного устройства и просматривать видео других пользователей в режиме реального времени.

Разработка мобильного приложения проводилась на языке программирования Kotlin в среде программирования Android Studio 4.1.1. Среда исполнения мобильного приложения Android 7.0 и выше.

Разработка серверной части проводилась на языке программирования Java с использованием Spring Boot в среде программирования IntelliJ IDEA 2020.3. Среда исполнения серверной части ОС Windows.

					ПГУ 09.03.04– 8БР171.13 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Маркичев К.А.		18.06.21				
Пров.		Князев В.Н.		18.06.21			3	127
Н. Контр.		Такташкин Д.В.		21.06.21		Группа 17ВП1		
Утв.		Макарычев П.П.						

Содержание

Введение.....	6
1 Анализ предметной области и требований.....	8
1.1 Анализ предметной области и постановка задач.....	8
1.2 Анализ решений аналогичного назначения	9
1.2.1 Анализ мобильного приложения «Omlet Arcade»	9
1.2.2 Анализ мобильного приложения «Streamlabs»	10
1.2.3 Анализ мобильного приложения «Twitch»	11
1.3 Анализ целевой аудитории	12
1.4 Анализ функциональных требований	15
1.5 Анализ нефункциональных требований	20
1.6 Планирование разработки и оценка бюджета.....	21
2 Проектирование мобильного приложения	35
2.1 Выбор технологий и инструментальных средств разработки.....	35
2.2 Проектирование архитектуры программного обеспечения	36
2.3 Проектирование пользовательского интерфейса	36
2.4 Проектирование серверной части приложения	43
2.4.1 Концептуальное и логическое проектирование базы данных.....	43
2.5 Проектирование серверной части приложения	46
3 Реализация мобильного приложения	47
3.1 Реализация серверной части	47
3.1.1 Реализация базы данных	47
3.1.2 Реализация серверной части приложения	47
3.2 Реализация клиентской части мобильного приложения.....	49
4 Тестирование программного продукта	52
4.1 Тестирование серверной части приложения	52
4.2 Тестирование клиентской части приложения.....	54
4.2.1 Функциональное тестирование.....	54
4.3 Метрики кода.....	56
4.3.1 SLOC-метрика	58

4.3.2 Метрики Холстеда	58
Заключение	61
Список использованных источников	62
Приложение А. Глоссарий	66
Приложение Б. Исходный код серверной части	68
Приложение В. Исходный код мобильного приложения	81
Приложение Г. Результаты тестирования серверной части	108
Приложение Д. Результаты тестирования мобильного приложения	113

Введение

В настоящее время информационные технологии используются почти во всех сферах человеческой жизни и являются наиболее быстро развивающейся отраслью. Им уделяется большое внимание не только в России, но и во всем мире. Об этом свидетельствует программа «Цифровая экономика Российской Федерации», утвержденная в 2017 году, основной задачей которой является «развитие цифровой экономики Российской Федерации, в которой данные в цифровой форме являются ключевым фактором производства во всех сферах экономической деятельности, что повышает конкурентоспособность страны, качество жизни граждан, обеспечивает экономический рост и экономический суверенитет» [1].

С увеличением количества владельцев смартфонов и постоянным усовершенствованием технологий возникает всё больше интернет сервисов и мобильных приложений. «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы» предусматривает широкомасштабное применение в информационном обществе web-сервисов сети «Интернет», а также различных мобильных устройств и мобильных приложений: «Электронные средства массовой информации, информационные системы, социальные сети, доступ к которым осуществляется с использованием сети «Интернет», стали частью повседневной жизни россиян. В России информационное общество характеризуется широким распространением и доступностью мобильных устройств» [2].

Сегодня большую популярность получили стриминговые платформы (Twitch, YouTube, GoodGame). Особенностью таких площадок стало их непосредственное направление. Стриминг образовано от слова стрим (stream – прямая трансляция). Множество людей, используя специальные программы, ведут прямые трансляции (стримы), предоставляя другим самый различный контент.

Основой работы данных платформ является работа с потоковым видео. Для приема и передачи потокового видео используются протоколы: RTMP (Real-Time Messaging Protocol), RTSP (Real-Time Streaming Protocol), SRT (Secure Reliable Transport Protocol) и др. [3].

В основном приложения для организации стримов делятся на приложения принимающие потоковое видео (Twitch, YouTube) и приложения, которые записывают видео с экрана и передают его на стриминговые платформы (OBS Studio, Wirecast, FFSplit) [4].

Однако данные приложения наиболее развиты для ПК. Мобильные устройства под управлением ОС Android предоставляют малое количество приложений для организации стримов. В то же время с увеличением производительности смартфонов, развиваются и игры для Android и iOS устройств.

Из выше сказанного можно сделать вывод об актуальности создания и развития мобильного приложения для организации стримов (просмотра и показа).

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задач

В последнее время стриминг в России очень сильно развивался как со стороны стримеров, так и со стороны зрителей. Наиболее показательными значениями в оценке развития стриминга является количество денежных перечислений в пользу стримеров, количество зрителей и количество часов просмотренного контента.

Так уже в 2018 году игроки, проводящие прямые трансляции, собрали 11,6 млрд руб. А количество зрителей, которые совершали переводы стримерам, превышало 11 млн человек.

В 2019 объем рынка стриминга в России и СНГ был менее 15 млрд руб., а в 2020 составил 21,6 млрд руб. (исследования QIWI), при этом средний чек доната (пожертвования) составил примерно 356 руб. [5].

В третьем квартале 2020 года люди посмотрели 4,7 млрд часов видео на Twitch. А всего на видеоплатформу было выгружено 206 млн часов контента.

Помимо прочего на развития рынка повлиял и период пандемии. В это время сильно увеличилось как количество зрителей, так и количество часов контента [6].

Вместе с тем неуклонно растет и производительность смартфонов. Для сравнения возьмем показатели синтетических тестов для самых топовых процессоров производителей Qualcomm, Apple, Samsung и HiSilicon. Рейтинг составлен на основании бенчмарка AnTuTu за 2019 и 2021 года. В таблице 1 представлен рейтинг за 2019 год [7] а в таблицу 2 – рейтинг за 2019 [8].

По данным из таблиц видно, что по результатам бенчмарка производительность процессоров, поставляемых для смартфонов, выросла почти в 2 раза.

Таблица 1 – Рейтинг AnTuTu за 2019 год

№	Процессор	Баллы
1	Snapdragon 855	368480
2	Apple A12	356501
3	Exynos 9820	325076
4	Kirin 980	309425

Таблица 2 – Рейтинг AnTuTu за 2019 год

№	Процессор	Баллы
1	Snapdragon 888	724190
2	Apple A14	615376
3	Exynos 2100	687795
4	Kirin 9000	693635

Следовательно, с развитием CPU смартфонов, развивается и рынок мобильных игр. Улучшается графика, левел-дизайн, усложняются игровые движки. Игр становится больше и к ним повышается интерес со стороны пользователей, что в конечном итоге приводит и к развитию стриминга на мобильной платформе.

1.2 Анализ решений аналогичного назначения

В настоящее время существуют аналогичные приложения, позволяющие записывать и/или просматривать потоковое видео. Проведем сравнительный анализ наиболее популярных программных продуктов данного сектора.

1.2.1 Анализ мобильного приложения «Omlet Arcade»

«Omlet Arcade» – приложение для записи и просмотра потокового видео, а также совместной игры. Данное приложение предоставляет возможность записывать видео как на свою площадку, так и другие популярные платформы. Главный экран мобильного приложения представлен на рисунке 1 [9].

Данное мобильное приложение можно считать лучшим среди платформ для стриминга. Приложение полностью на русском языке. Не смотря на, местами перегруженный интерфейс, приложение является достаточно понятным с точки зрения управления для пользователя. Также достоинством является возможность передачи потокового видео на другие платформы.

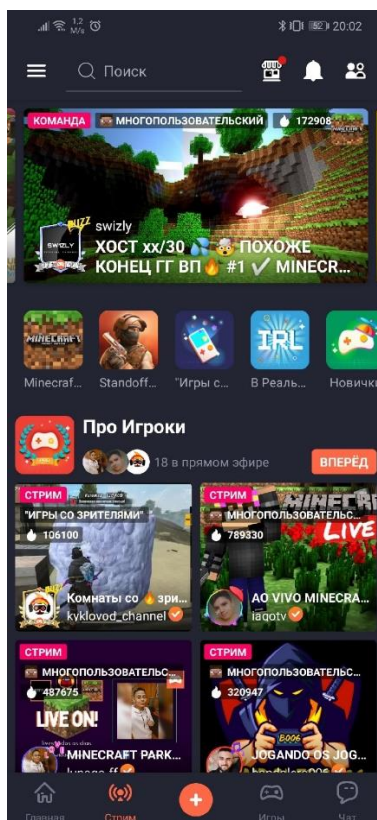


Рисунок 1 – Главный экран мобильного приложения «Omlet Arcade»

Однако, приложение имеет весомый недостаток, а именно возможность передачи потокового видео в хорошем качестве (от 1080p) только в платной премиум подписке.

1.2.2 Анализ мобильного приложения «Streamlabs»

«Streamlabs» – приложение, позволяющее записывать и передавать на другие платформы потоковое видео. Также приложение поддерживает передачу по собственной RTMP ссылке. Авторизация в приложении доступна через такие популярные платформы как: Twitch, YouTube и Facebook. Экран с виджетами, выводимыми в приложении, представлен на рисунке 2 [10].

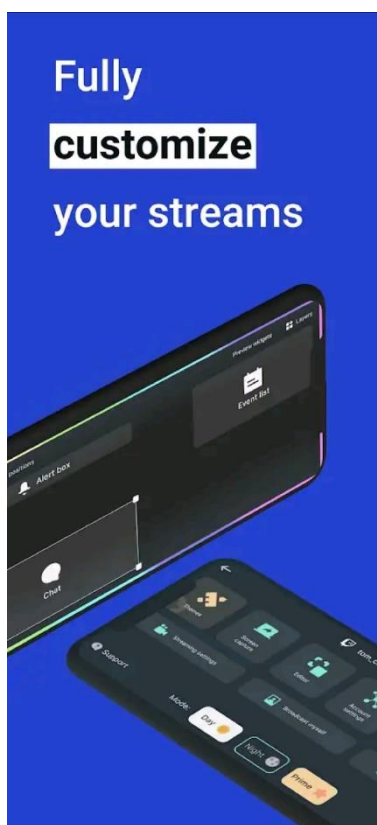


Рисунок 2 – Экран с виджетами мобильного приложения «Streamlabs»

Главным преимуществом приложения является возможность вывода различных уведомлений (сообщения чата, донаты) в виде виджетов во время записи игрового контента.

Из недостатков можно выделить отсутствие возможности просмотра трансляций других пользователей, а также русского языка, что осложняет взаимодействие пользователя с интерфейсом.

1.2.3 Анализ мобильного приложения «Twitch»

«Twitch» – видеоплатформа, позволяющая смотреть трансляции игр и событий. Данная платформа является ведущей в мире, что можно сразу отнести к плюсам. На ней транслируются главные события киберспортивного мира. Главный экран приложения представлен на рисунке 3 [11].

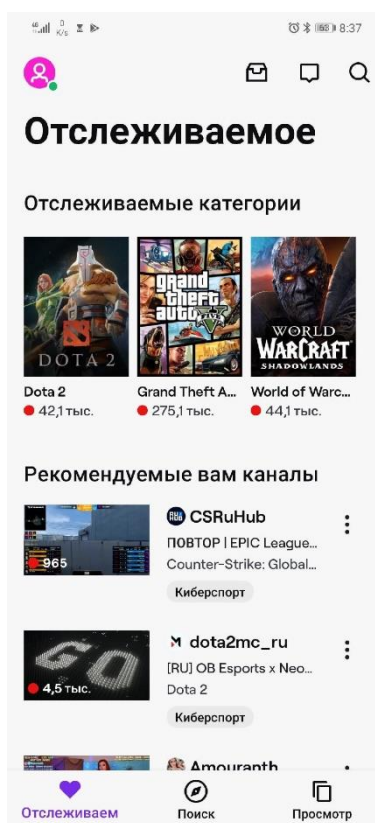


Рисунок 3 – Главный экран мобильного приложения «Twitch»

Основными преимуществами, помимо популярности площадки, является поддержка русского языка и понятный интерфейс. А из недостатков – отсутствие у данной платформы возможности записи трансляций (для проведения прямых трансляций приходится использовать сторонние приложения).

Таким образом, по результатам анализа программ-аналогов, было принято решение о создании собственного мобильного приложения организации записи и просмотра игрового контента с использованием потокового видео, которое не должно содержать выделенных у данных продуктов недостатков.

1.3 Анализ целевой аудитории

Целевой аудиторией для разрабатываемого приложения являются пользователи, записывающие игровой процесс, для трансляции его в прямом эфире, и просматривающие прямые эфиры других пользователей.

На рисунке 4 представлены доли мирового рынка мобильных операционных систем. Из графика видно, что наибольшее распространение в мире имеют мобильные устройства с операционной системой Android (более 65%). Второе место занимают устройства с ОС iOS, а на третьем остальные операционные системы [12].

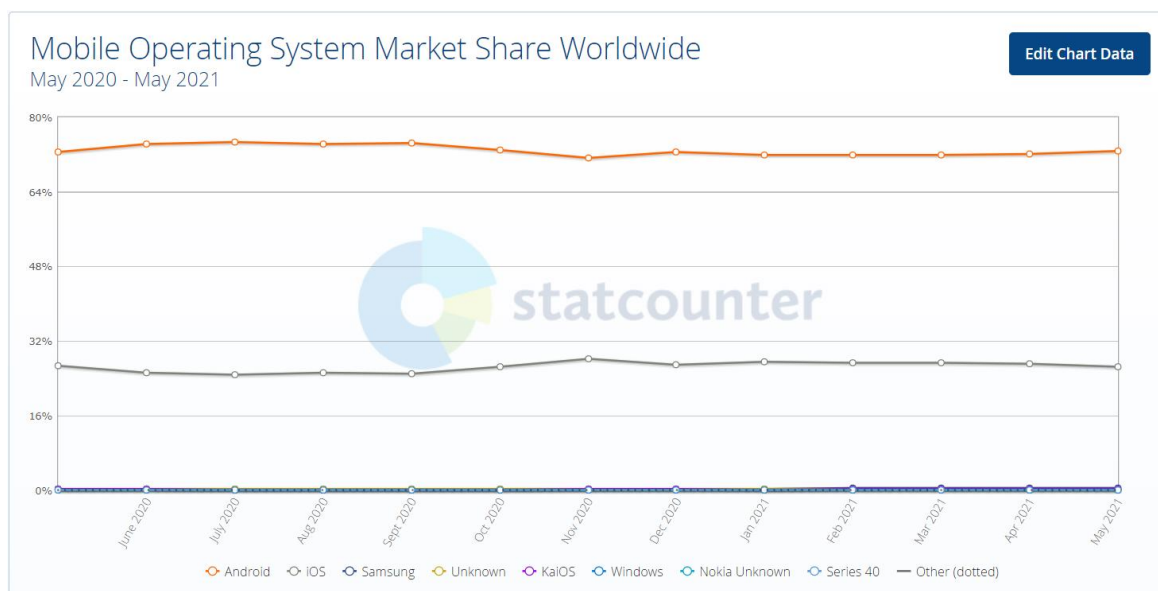


Рисунок 4 – Доли рынка мобильных операционных систем

На рисунке 5 представлены доли производителей смартфонов на российском рынке. Лидеры этого списка – Xiaomi, Samsung и Honor являются мобильными устройствами, работающими под управлением ОС Android [13].



Рисунок 5 – Доли производителей смартфонов на российском рынке

Данная статистика показывает целесообразность выбора операционной системы, для которой будет разрабатываться мобильное приложение.

На рисунке 6 представлен охват устройств в процентах в зависимости от выбора минимально поддерживаемой приложением версии Android. Данное распределение было получено непосредственно в официальной IDE Android Studio, предназначенной для разработки Android приложений.

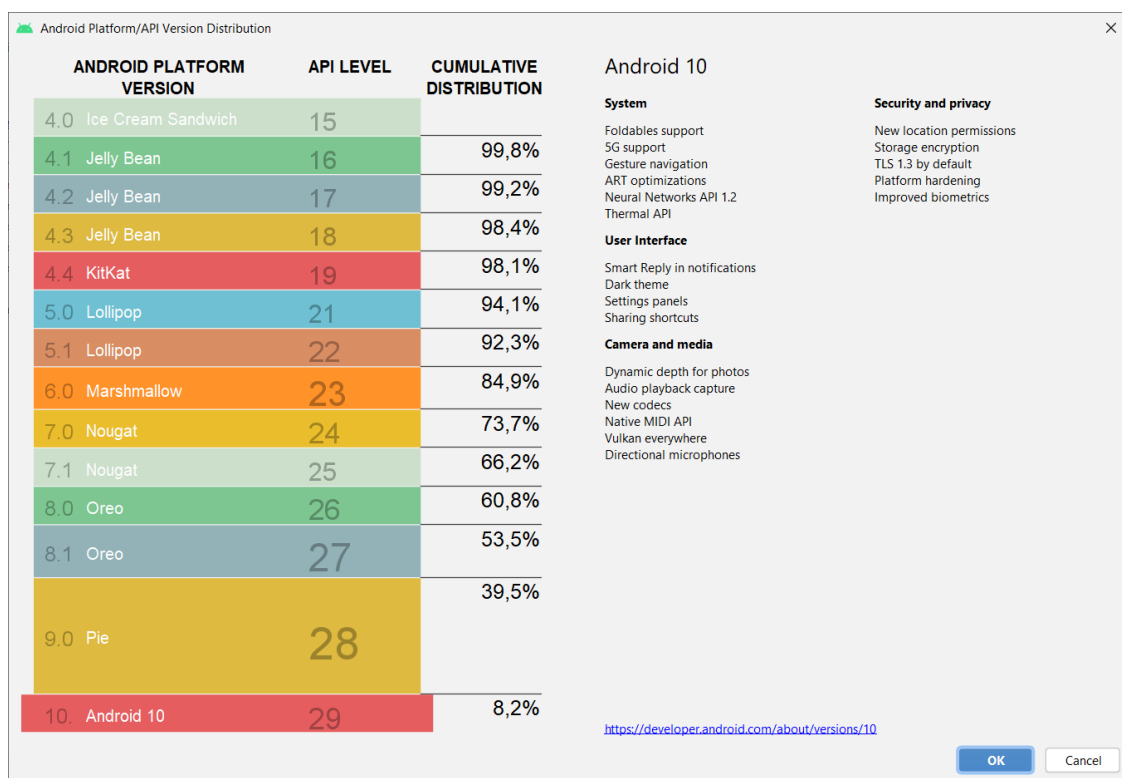


Рисунок 6 – Охват устройств в процентах

При выборе минимально поддерживаемой приложением версии Android стоит также учесть, что версии ниже 9.0 уже считаются устаревшими. Однако охват устройств приложением с минимальной версией 9.0 будет слишком маленьким. Версия 6.0 была выпущена и распространялась на устройства в конце 2015 – 2016 годах [14]. Данные устройства могут не соответствовать с аппаратной точки зрения. Таким образом, можно сделать вывод, что наиболее оптимальным выбором для разрабатываемого приложения будет поддержка минимальной версии Android 7.0 «Nougat».

1.4 Анализ функциональных требований

Пользователи приложения будут разделены на два типа: авторизованные пользователи и не авторизованные. Авторизованным пользователям будет доступен весь функционал приложения. А пользователи, не прошедшие процесс авторизации, не будут иметь доступ к профилю, подпискам, уведомлениям, чатам в прямых трансляциях, а также возможность запускать собственные трансляции.

В результате анализа предметной области и аналогичных программных продуктов были выделены функциональные требования к программе [15]. Для не авторизованных пользователей были выделены следующие основные функции:

- зарегистрироваться в приложении;
- выполнить вход в приложении;
- посмотреть список текущих трансляций;
- выбрать трансляцию.

Для авторизованных пользователей были выделены следующие основные функции:

- посмотреть список текущих трансляций;
- выбрать трансляцию;
- написать сообщение в чат;
- подписаться на другого пользователя;
- отписаться от пользователя;
- подписаться на игру;
- отписаться от игры;
- начать собственную трансляцию;
- закончить трансляцию;
- посмотреть профиль;
- посмотреть подписчиков;
- редактировать данные профиля;

- посмотреть список подписок;
- посмотреть уведомления.

Диаграммы вариантов использования представлены на рисунке 7 и 8.

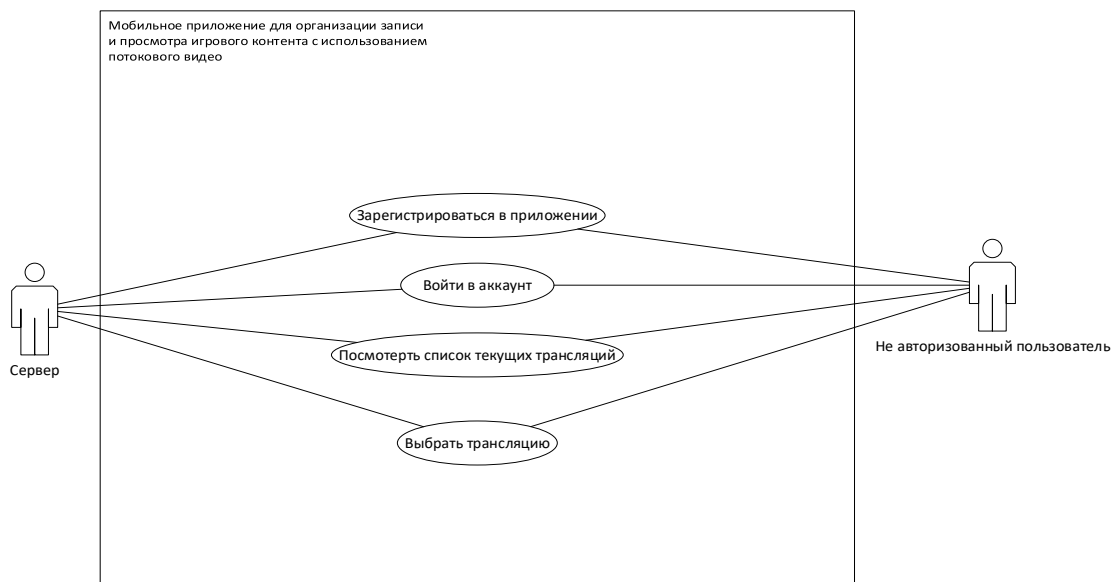


Рисунок 7 – Диаграмма вариантов использования для неавторизованного пользователя



Рисунок 8 – Диаграмма вариантов для авторизованного пользователя

Рассмотрим некоторые прецеденты более подробно, для этого приведем их подробную спецификацию.

На рисунке 9 описана спецификация для варианта использования «Начать собственную трансляцию», который доступен только для авторизованных пользователей. На рисунке 10 описана спецификация для варианта использования «Войти в аккаунт», который необходим для перехода от не авторизованного пользователя к авторизованному пользователю. На рисунке 11 описана спецификация для варианта использования «Подписаться на другого пользователя» [16].

Прецедент: Начать собственную трансляцию
ID: 1
Краткое описание: Пользователь заполняет данные о трансляции и начинает записывать видео с экрана
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Сервер
Предусловия: Пользователь должен быть авторизован
Основной поток: <ol style="list-style-type: none"> 1. Пользователь вводит название трансляции 2. Пользователь выбирает обложку трансляции 3. Пользователь выбирает игру, которая будет транслироваться 4. Пользователь запускает трансляцию 5. Приложение передаёт записываемое видео на сервер
Постусловия: В списке трансляций отображается новая трансляция, запущенная текущим пользователем
Альтернативные потоки: нет

Рисунок 9 – Спецификация варианта использования «Начать собственную трансляцию»

Прецедент: Войти в аккаунт
ID: 2
Краткое описание: Не авторизованный пользователь заполняет данные для аутентификации
Главные актеры: Не авторизованный пользователь
Второстепенные актеры: Сервер
Предусловия: Пользователь должен быть не авторизован
Основной поток: <ol style="list-style-type: none"> 1. Пользователь вводит данные (Логин, Пароль) 2. Пользователь нажимает «Войти» 3. Сервер возвращает данные пользователя 4. Пользователь авторизован
Постусловия: Отображается профиль пользователя
Альтернативные потоки: нет

Рисунок 10 – Спецификация варианта использования «Аутентифицироваться в приложении»

Прецедент: Подписаться на другого пользователя
ID: 3
Краткое описание: Авторизованный пользователь подписывается на другого пользователя
Главные актеры: Авторизованный пользователь
Второстепенные актеры: Сервер
Предусловия: Пользователь должен быть авторизован
Основной поток: <ol style="list-style-type: none"> 1. Пользователь переходит на подробную информацию о другом пользователе 2. Пользователь нажимает «Подписаться» 3. Сервер обновляет данные текущего пользователя
Постусловия: В подписках пользователя отображается пользователь, на которого он подписался
Альтернативные потоки: нет

Рисунок 11 – Спецификация варианта использования «Подписаться на другого пользователя»

Для некоторых вариантов использования составим диаграммы деятельности.

На рисунке 12 приведена диаграмма деятельности для прецедента «Редактировать данные профиля», который доступен только для авторизованного пользователя. В рамках данного прецедента проверяются введенные пользователем данные, сначала на корректность (используемые символы и их количество), а затем использование этих данных другими пользователями.

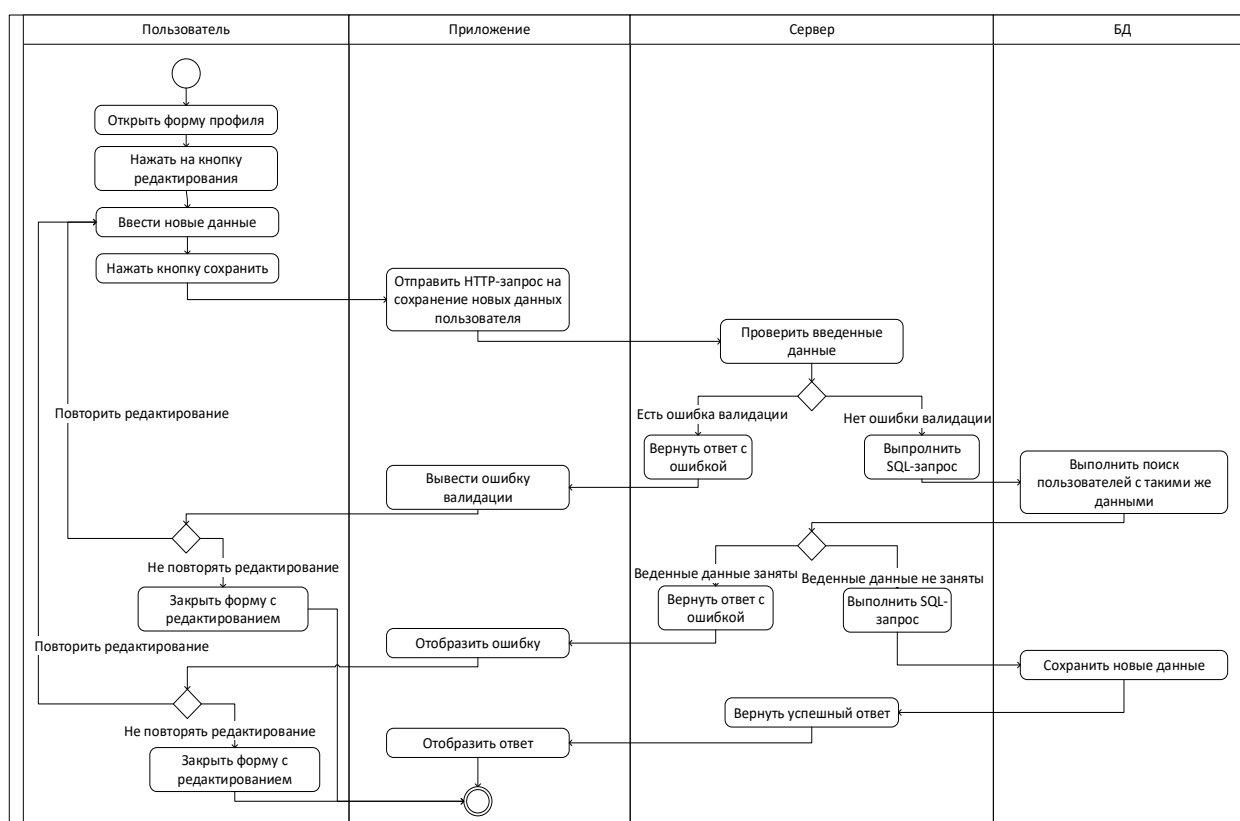


Рисунок 12 – Диаграмма деятельности для варианта использования «Редактировать данные профиля»

На рисунке 13 приведена диаграмма деятельности для прецедента «Посмотреть список текущих трансляций». Данный прецедент доступен для обоих типов пользователей.

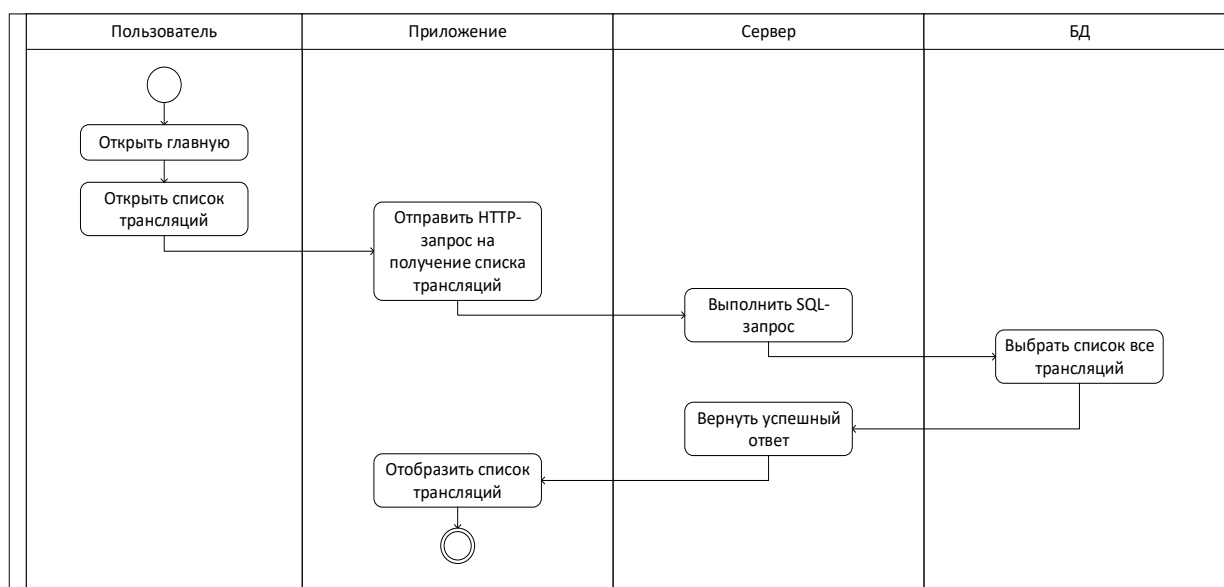


Рисунок 13 – Диаграмма деятельности для варианта использования
«Посмотреть список текущих трансляций»

В результате анализа функциональных требований, были уточнены и определены функции, которые необходимо реализовать в разрабатываемом приложении, были разработаны диаграммы вариантов использования, диаграммы деятельности.

1.5 Анализ нефункциональных требований

Разработка серверной части приложения должна быть выполнена на языке программирования Java с использованием Spring Boot, в среде программирования IntelliJ IDEA 2020. Серверная часть должна включать в себя API (интерфейс прикладного программирования). Клиентская часть приложения должна быть реализована на языке программирования Kotlin. В качестве целевой СУБД должна выступать PostgreSQL, запросы к базе данных должны осуществляться при помощи языка SQL.

Клиентская часть приложения должна иметь удобный и интуитивно понятный интерфейс, клиентская часть должна быть полностью отлажена на предмет возникновения исключительных ситуаций. Мобильное приложение

должно исполняться на мобильных устройствах под управлением ОС Android 7.0 и выше.

1.6 Планирование разработки и оценка бюджета

Структурное планирование можно разделить на следующие этапы:

- разбиение проекта на совокупность отдельных работ, выполнение которых необходимо для реализации проекта;
- построение сетевого графика, описывающего последовательность выполнения работ;
- оценка временных характеристик работ и анализ сетевого графика.

На этапе структурного планирования основную роль играет сетевой график. Он представляет собой ориентированный граф, в котором вершины – работы проекта, а дуги – временные взаимосвязи работ [17].

Сетевой график должен удовлетворять следующим свойствам:

- каждой работе соответствует только одна вершина, т.е. ни одна работа не может быть представлена на сетевом графике дважды;
- ни одна работа не может быть начата, пока не закончатся все непосредственно предшествующие ей работы (предшествующие работы – работы, дуги которых входят в данную работу);
- ни одна работа, которая непосредственно следует за некоторой работой, не может начаться до момента ее окончания;
- начало и конец проекта обозначены работами с нулевой продолжительностью или вехами.

Сетевой график позволяет определить критические работы проекта и его критический путь.

Работа, для которой задержка ее начала приведет к задержке срока окончания проекта в целом является критической. Данные работы не имеют запаса времени. Работа, имеющая некоторый запас времени, в пределах которого их начало может быть задержано, называется некритической.

Критический путь – это путь от начальной к конечной вершине сетевого графика, проходящий только через критические работы. Суммарная длительность работ критического пути определяет минимальное время реализации проекта.

Определения критического пути сводится к нахождению критических работ проекта [18]. Это выполняется в два этапа:

- вычисление раннего времени начала каждой работы проекта (время раньше которого работы не может быть начата);
- вычисление позднего времени каждой работы (время, позже которого не может быть начата работа без увеличения продолжительности всего проекта).

Раннее и позднее время критических работ совпадает.

Пусть $T_p(i)$ – раннее время начала работы i , $T_n(i)$ – позднее время начала работы i , t_j – время выполнения работы j . Тогда

$$T_p(i) = \max(T_p(j) + t_j) \quad j \in G,$$

$$T_n(i) = \min(T_n(j) - t_j) \quad j \in H,$$

где G – множество работ, непосредственно предшествующих работе i ;

H – это множество работ, непосредственно следующих за работой i .

Раннее время начальной работы проекта принимается равным нулю.

Так как последняя работа – это веха с нулевой длительностью, то раннее время ее начала совпадает с длительностью всего проекта.

Определим критические работы и критический путь для программного проекта разработки мобильного приложения для организации записи и просмотра игрового контента с использованием потокового видео.

Правильное планирование позволит оценить стоимость и трудоемкость проекта. Для чего были определены задачи и ресурсы, которые необходимы для создания мобильного приложения.

В таблице 3 представлены задачи, которые представляют этапы создания проекта.

Таблица 3 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	-	-
2	Анализ требований	-	-
3	Анализ предметной области	7 дней	1
4	Анализ аналогичных решений	4 дней	3
5	Анализ целевой аудитории	4 дня	4
6	Принятие окончательного решения	1 день	5
7	Анализ требований завершен	-	6
8	Выбор средств реализации и архитектуры проекта	-	-
9	Анализ возможных программных средств	4 дня	1
10	Анализ архитектур, используемых в разработке Android приложений	3 дня	9
11	Анализ архитектур, используемых при разработке серверных приложений	3 дня	10
12	Выбор средств и архитектуры завершен	-	11
13	Реализация проекта	-	-
14	Разработка структуры данных	1 день	7; 12
15	Разработка пользовательского интерфейса	10 дней	14
16	Разработка мобильного приложения	20 дней	15
17	Разработка серверной части приложения	25 дней	14
18	Реализация проекта завершена	-	16; 17
19	Отладка	-	-
20	Тестирование мобильной части приложения	5 дней	16
21	Тестирование серверной части приложения	5 дней	17
22	Тестирование совместной работы мобильного приложения и серверной части	5 дней	18
23	Исправление ошибок	10 дней	20; 21; 22
24	Тестирование завершено	-	23

Продолжение таблицы 3

№	Название задачи	Длительность	Предшественники
25	Публикация приложения в Google Play	-	-
26	Создание аккаунта в Google Play Developer Console	1 день	24
27	Подготовка маркетинговых материалов (иконка, скриншоты, APK, текст, баннер, проморолик)	5 дней	24
28	Отправка сборки в Google Play	1 день	26; 27
29	Публикация завершена	-	28
30	Конец проекта	-	29

Из таблицы, представленной выше видно, что для реализации проекта необходимо выполнить 30 задач.

В таблице 4 представлено описание затрачиваемых ресурсов.

Таблица 4 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист1	Трудовой	250 Р/ч	300 Р/ч
Программист2	Трудовой	200 Р/ч	250 Р/ч
Программист3	Трудовой	150 Р/ч	200 Р/ч
Дизайнер	Трудовой	150 Р/ч	200 Р/ч
Интернет	Затраты		
Компьютер	Материальный	500 Р	

Распределение ресурсов по задачам представлено в таблице 5.

Таблица 5 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта		
2	Анализ требований		
3	Анализ предметной области	Программист1 Компьютер Интернет	60% 1 шт
4	Анализ аналогичных решений	Программист1 Программист2 Программист3 Компьютер	30% 35% 35% 1 шт
5	Анализ целевой аудитории	Программист1 Компьютер	50% 1 шт

Продолжение таблицы 5

№	Название задачи	Ресурс	Единицы
6	Принятие окончательного решения	Программист1 Программист2 Программист3	100% 100% 100%
7	Анализ требований завершен		
8	Выбор средств реализации и архитектуры проекта		
9	Анализ возможных программных средств	Программист1 Компьютер Интернет	40% 1 шт
10	Анализ архитектур, используемых в разработке Android приложений	Программист1 Программист2 Компьютер Интернет	50% 50% 1 шт
11	Анализ архитектур, используемых при разработке серверных приложений	Программист1 Программист3 Компьютер Интернет	50% 50% 1 шт
12	Выбор средств и архитектуры завершен		
13	Реализация проекта		
14	Разработка структуры данных	Программист1 Программист2 Компьютер Интернет	75% 25% 1 шт
15	Разработка пользовательского интерфейса	Дизайнер Компьютер Интернет	100% 1 шт
16	Разработка мобильного приложения	Программист1 Программист2 Компьютер Интернет	10% 90% 1 шт
17	Разработка серверной части приложения	Программист1 Программист3 Компьютер Интернет	10% 90% 1 шт
18	Реализация проекта завершена		
19	Отладка		
20	Тестирование мобильной части приложения	Программист1 Программист2 Дизайнер Компьютер Интернет	25% 75% 75% 2 шт

Окончание таблицы 5

№	Название задачи	Ресурс	Единицы
21	Тестирование серверной части приложения	Программист1 Программист3 Компьютер Интернет	25% 75% 1 шт
22	Тестирование совместной работы мобильного приложения и серверной части	Программист1 Программист2 Программист3 Компьютер Интернет	50% 50% 50% 3 шт
23	Исправление ошибок	Программист1 Программист2 Программист3 Компьютер Интернет	30% 100% 100% 3 шт
24	Тестирование завершено		
25	Публикация приложения в Google Play		
26	Создание аккаунта в Google Play Developer Console	Программист1 Компьютер Интернет	90% 1 шт
27	Подготовка маркетинговых материалов (иконка, скриншоты, APK, текст, баннер, проморолик)	Программист1 Дизайнер Компьютер Интернет	10% 90% 1 шт
28	Отправка сборки в Google Play	Программист1 Компьютер Интернет	100% 1 шт
29	Публикация завершена		
30	Конец проекта		

Из представленных таблиц видно, что реализацией проекта будут заниматься 3 программиста и 1 дизайнер, которым для работы нужны компьютеры и доступ к интернету, а также распределение программистов и дизайнера между задачами.

Для вычисления длительности реализации проекта и его критического пути построим сетевой график. Он представлен на рисунке 14.

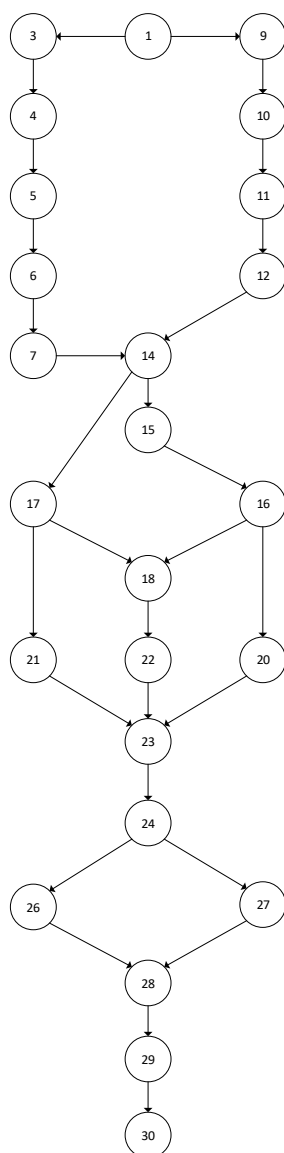


Рисунок 14 – Сетевой график проекта

Исходя из полученного сетевого графика и длительности задач из таблицы 1 вручную с использованием соответствующих приведенных ранее формул рассчитаем критический путь проекта. В таблице 6 представлены расчеты раннего времени начала работ каждой задачи.

Таблица 6 – Расчеты раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	-	-	-	-	0
3	1	0	0	0+0	0

Продолжение таблицы 6

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
4	3	0	7	0+7	7
5	4	7	4	7+4	11
6	5	11	4	11+4	15
7	6	15	1	15+1	16
9	1	0	0	0+0	0
10	9	0	4	0+4	4
11	10	4	3	4+3	7
12	11	7	3	7+3	10
14	7	16	0	Max(16+0;10+0)	16
	12	10	0		
15	14	16	1	16+1	17
16	15	17	10	17+10	27
17	14	16	1	16+1	17
18	16	27	20	Max(27+20;17+25)	47
	17	17	25		
20	16	27	20	27+20	47
21	17	17	25	17+25	42
22	18	47	0	47+0	47
23	20	47	5	Max(47+5;42+5)	52
	21	42	5		
	22	47	5		
24	23	52	10	52+10	62
26	24	62	0	62+0	62
27	24	62	0	62+0	62
28	26	62	1	Max(62+1;62+5)	67
	27	62	5		
29	28	67	1	67+1	68
30	29	68	0	68+0	68

В таблице 7 представлены расчеты позднего времени начала работ каждой задачи.

Таблица 7 – Расчеты позднего времени начала работ

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
29	30	68	0	68-0	68
28	29	68	1	68-1	67

Продолжение таблицы 7

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
27	28	67	5	67-5	62
26	28	67	1	67-1	66
24	27	62	0	Min(62-0;66-0)	62
	26	66			
23	24	62	10	62-10	52
22	23	52	5	52-5	47
21	23	52	5	52-5	47
20	23	52	5	52-5	47
18	22	47	0	47-0	47
17	21	47	25	Min(47-25;47-25)	22
	18	47			
16	20	47	20	Min(47-20;47-20)	27
	18	47			
15	16	27	10	27-10	17
14	17	22	1	Min(22-1;17-1)	16
	15	17			
12	14	16	0	16-0	16
11	12	16	3	16-3	13
10	11	13	3	13-3	10
9	10	10	4	10-4	6
7	14	16	0	16-0	16
6	7	16	1	16-1	15
5	6	15	4	15-4	11
4	5	11	4	11-4	7
3	4	7	7	7-7	0
1	9	6	0	Min(6-0;0-0)	0
	3	0			

Исходя из найденных значений раннего и позднего начала работ можно определить работы, через которые проходит критический путь. Все вычисления представлены в таблице 8.

Таблица 8 – Вычисление критического пути

Работа №	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	0	0
4	7	7	0
5	11	11	0
6	15	15	0
7	16	16	0
9	0	6	6
10	4	10	6
11	7	13	6
12	10	16	6
14	16	16	0
15	17	17	0
16	27	27	0
17	17	22	5
18	47	47	0
20	47	47	0
21	42	47	5
22	47	47	0
23	52	52	0
24	62	62	0
26	62	66	4
27	62	62	0
28	67	67	0
29	68	68	0
30	68	68	0

Из таблицы, полученной в результате вычислений, можно сделать вывод, что критический путь проходит через работы с номерами 1, 3, 4, 5, 6, 7, 14, 15, 16, 18, 20, 22, 23, 24, 27, 28, 29, 30. А длительность реализации проекта составляет 68 дней. На рисунке 15 представлен сетевой граф с указанием критического пути.

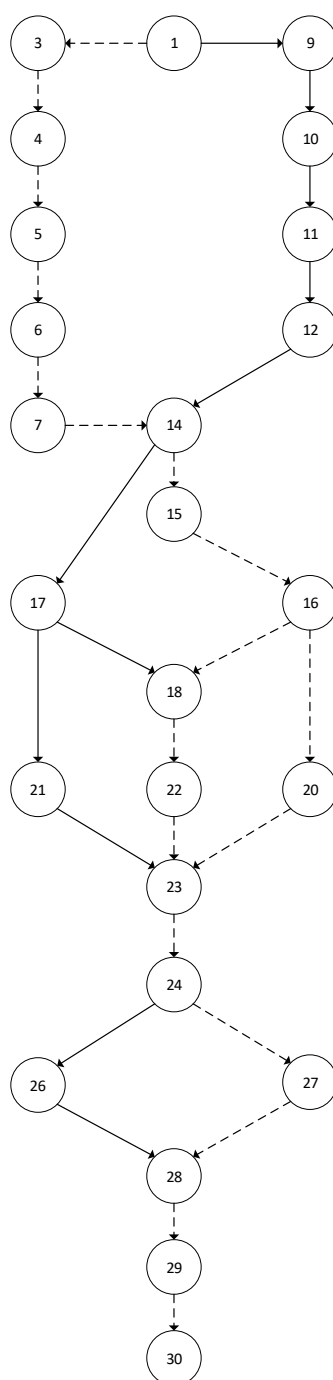


Рисунок 15 – Критический путь проекта

Проведем необходимые вычисления с использованием средства управления программными проектами Microsoft Project 2016.

На рисунке 16 представлен список задач из таблицы 1 внесенный в Microsoft Project. Началом проекта была принята дата 01.03.2021 [19].

1	Начало проекта	0 дней	Пн 01.03.21	Пн 01.03.21	
2	Анализ требований	16 дней	Пн 01.03.21	Пн 22.03.21	
3	Анализ предметной области	7 дней	Пн 01.03.21	Вт 09.03.21	1
4	Анализ аналогичных решений	4 дней	Ср 10.03.21	Пн 15.03.21	3
5	Анализ целевой аудитории	4 дней	Вт 16.03.21	Пт 19.03.21	4
6	Принятие окончательного решения	1 день	Пн 22.03.21	Пн 22.03.21	5
7	Анализ требований завершен	0 дней	Пн 22.03.21	Пн 22.03.21	6
8	Выбор средств реализации и архитектуры проекта	10 дней	Пн 01.03.21	Пт 12.03.21	
9	Анализ возможных программных средств	4 дней	Пн 01.03.21	Чт 04.03.21	1
10	Анализ архитектур, используемых в разработке Android приложений	3 дней	Пт 05.03.21	Вт 09.03.21	9
11	Анализ архитектур, используемых при разработке серверных приложений	3 дней	Ср 10.03.21	Пт 12.03.21	10
12	Выбор средств и архитектур завершен	0 дней	Пт 12.03.21	Пт 12.03.21	11
13	Реализация проекта	31 дней	Вт 23.03.21	Вт 04.05.21	
14	Разработка структуры данных	1 день	Вт 23.03.21	Вт 23.03.21	7;12
15	Разработка пользовательского интерфейса	10 дней	Ср 24.03.21	Вт 06.04.21	14
16	Разработка мобильного приложения	20 дней	Ср 07.04.21	Вт 04.05.21	15
17	Разработка серверной части приложения	25 дней	Ср 24.03.21	Вт 27.04.21	14
18	Реализация проекта завершена	0 дней	Вт 04.05.21	Вт 04.05.21	16;17
19	Отладка	20 дней	Ср 28.04.21	Вт 25.05.21	
20	Тестирование мобильной части приложения	5 дней	Ср 05.05.21	Вт 11.05.21	16
21	Тестирование серверной части приложения	5 дней	Ср 28.04.21	Вт 04.05.21	17
22	Тестирование совместной работы мобильного приложения и серверной части	5 дней	Ср 05.05.21	Вт 11.05.21	18
23	Исправление ошибок	10 дней	Ср 12.05.21	Вт 25.05.21	20;21;22
24	Тестирование завершено	0 дней	Вт 25.05.21	Вт 25.05.21	23
25	Публикация приложения в Google Play	6 дней	Ср 26.05.21	Ср 02.06.21	
26	Создание аккаунта Google Play Developer Console	1 день	Ср 26.05.21	Ср 26.05.21	24
27	Подготовка маркетинговых материалов	5 дней	Ср 26.05.21	Вт 01.06.21	24
28	Отправка сборки в Google Play	1 день	Ср 02.06.21	Ср 02.06.21	26;27
29	Публикация завершена	0 дней	Ср 02.06.21	Ср 02.06.21	28
30	Конец проекта	0 дней	Ср 02.06.21	Ср 02.06.21	29

Рисунок 16 – Задачи проекта в среде Microsoft Project

На рисунке 17 представлена диаграмма Ганта, которая была автоматически построена в среде Microsoft Project на основе введенных данных.

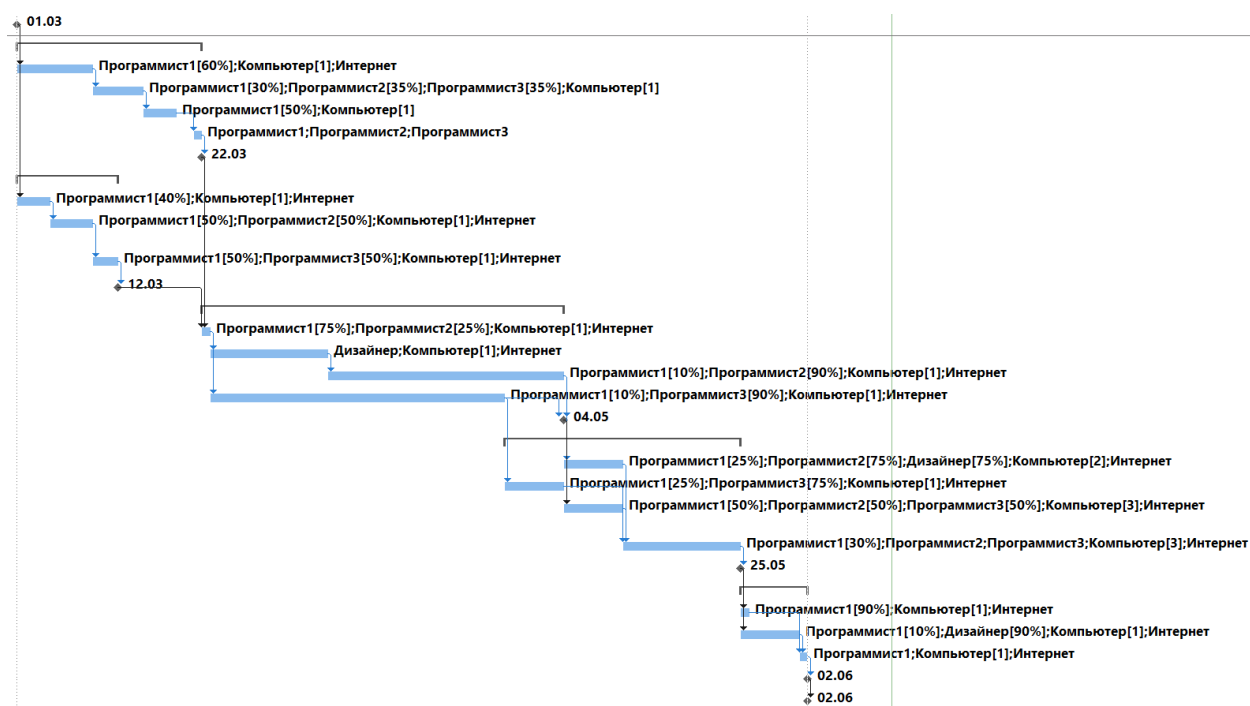


Рисунок 17 – Диаграмма Ганта

На рисунке 18 представлен обзор проекта. Из рисунка видно, что общая длительность проекта составляет 68 дней, при этом предполагаемые затраты – 198 140,00 руб. Длительность реализации проекта, вычисленная в программе, совпадает со значением, вычисленным вручную.

Мобильное приложение для организации записи и просмотра игрового контента с использованием потокового видео

Начало		Окончание	
Пн 01.03.21		Ср 02.06.21	
Длительность	Оставшаяся длительность	% завершения	
68 дней	68 дней	0%	
Трудозатраты	Сверхурочные трудозатраты	Фактические трудозатраты	
996,4 ч	0 ч	0 ч	
Затраты	Затраты на сверхурочные	Оставшиеся затраты	Фактические затраты
198 140,00 ₽	0,00 ₽	198 140,00 ₽	0,00 ₽

Рисунок 18 – Обзорный отчет проекта

Таким образом, было проведено грамотное планирование проекта, а также оценена его стоимость. Полученные результаты свидетельствуют об экономической целесообразности данного проекта.

2 Проектирование мобильного приложения

2.1 Выбор технологий и инструментальных средств разработки

На сегодняшний день существует достаточно много методик по разработке ПО. Данные методики различаются используемой моделью жизненного цикла (ЖЦ) ПО.

Классический ЖЦ определяется каскадной или водопадной моделями. В данных моделях предполагается последовательное выполнение этапов создания ПО. При этом не предполагается возврат на предыдущие этапы. Такие методики применимы к простым программным системам, когда мало неопределенностей, а используемые технологические решения известны и проверены.

Еще одной моделью ЖЦ является спиральная модель. Данная модель повторение этапов: анализ, проектирование, разработка, тестирование и сопровождение. Однако проблемой модели является определение момента перехода на следующий этап. Спиральная модель лежит в основе методологии Microsoft Solution Framework (MSF).

Наиболее подходящей для данного проекта является инкрементная модель ЖЦ. Модель предполагает разбиение проекта на более мелкие и итеративное наращивание функционала системы. По окончании каждой итерации получается функционирующий программный продукт. Модель легла в основу методологии Rational Unified Process (RUP). Данная методология может быть адаптирован под разработку различных программных систем [20].

Еще одним преимуществом RUP является использование языка визуального моделирования Unified Modeling Language (UML). UML – унифицированный графический язык для объектного моделирования в области разработки программных продуктов. Он используется для моделирования программного обеспечения и бизнес-процессов, системного проектирования и отображения организационных структур [21].

Мобильное приложение разрабатывалось на языке программирования Kotlin, который рекомендуется Google для создания приложений для Android. Для разработки мобильного приложения использовалась среда разработки Android Studio.

Серверная часть приложения разрабатывалась на языке программирования Java с использованием библиотеки Spring Boot в среде разработки IntelliJ IDEA 2020. Для хранения данных использовалась СУБД PostgreSQL, языком запросов для базы данных был выбран SQL.

2.2 Проектирование архитектуры программного обеспечения

При разработке мобильного приложения была выбрана двухуровневая архитектура – клиент-сервер. Выбор данной архитектуры был сделан исходя из того, что приём, обработка и передача остальным пользователям потокового видео является ресурсоёмким процессом и не должна выполняться на стороне клиента. Помимо этого, приложение будет хранить в себе большой объем данных, который лучше всего будет перенести на сервер.

При разработке клиентской части был выбран паттерн MVVM, позволяющий разделить логику представления от бизнес-логики. Разделение приложения на три компонента (Model-View-ViewModel) позволит в дальнейшем проще модифицировать и поддерживать приложение, а также упростить тестирование [22].

2.3 Проектирование пользовательского интерфейса

На основе выявленных функциональных требований был спроектирован пользовательский интерфейс мобильного приложения. К основным формам приложения относятся следующие: главная форма, форма трансляции, форма завершения трансляции, форма с подписками, форма начала трансляции, форма с уведомлениями, форма профиля. Также были выделены диалоговые формы: форма авторизации, форма регистрации, форма редактирования профиля.

На рисунке 19 представлена главная форма приложения.

Цифрами на рисунке отмечены следующие компоненты формы:
1 – список трансляций, 2 – список стримеров, 3 – нижнее меню навигации.

По нажатию на элемент списка трансляций будет выполнен переход на форму трансляции. Нижнее меню навигации позволяет перемещаться между следующими формами приложения: главная форма, форма с подписками, форма начала трансляции, форма уведомлений и форма профиля.

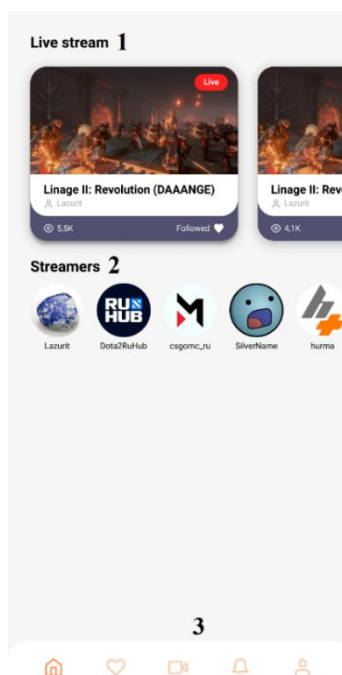


Рисунок 19 – Главная форма

На рисунке 20 представлена форма трансляции, на которой расположены следующие элементы: 1 – проигрыватель видео, 2 – информация о трансляции (название трансляции, количество зрителей и время начала трансляции), 3 – информация о стримере (название канала, иконка канала и количество подписчиков), 4 – кнопка «Подписаться», 5 – чат, 6 – кнопка для добавления сообщения в чат.

По нажатию на кнопку «Подписаться» стример, ведущий трансляцию, будет добавлен в список подписок пользователя, а текст кнопки будет изменен на «Отписаться».

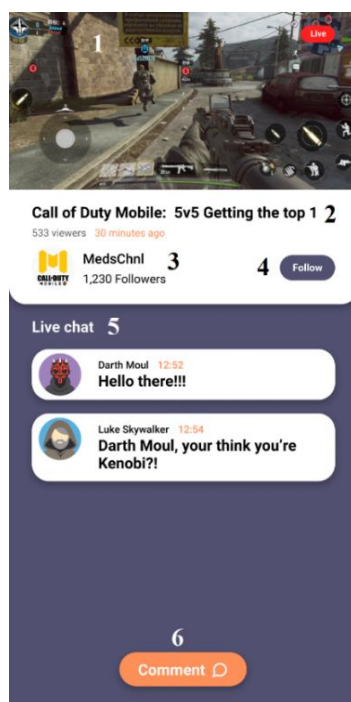


Рисунок 20 – Форма трансляции

На рисунке 21 представлена форма с подписками, состоящая из следующих компонентов: 1 – список стримеров, 2 – кнопка для перехода на форму со всеми стримерами, на которых подписан пользователь, 3 – список избранных игр, 4 – кнопка для перехода на форму всех избранных игр.

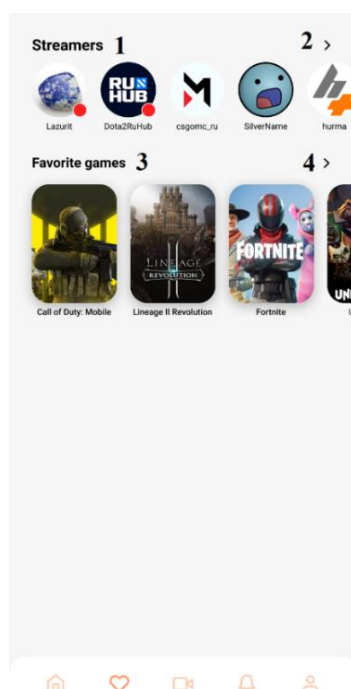


Рисунок 21 – Форма с подписками

На рисунке 22 представлена форма для начала трансляции, состоящая из следующих элементов: 1 – текстовое поле, для ввода пользователем названия трансляции, 2 – список игр.

Название трансляции является обязательным полем для ввода. Начало трансляции осуществляется по нажатию на игру.

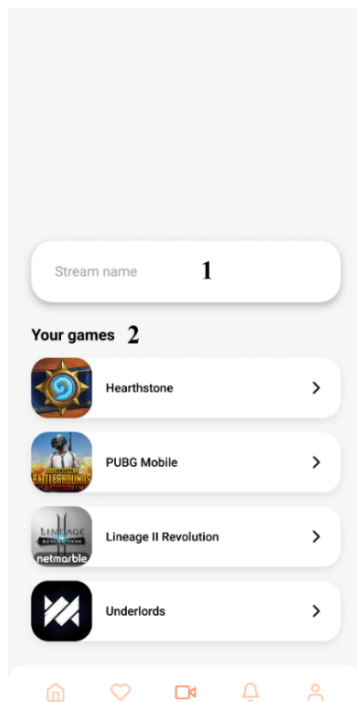


Рисунок 22 – Форма для начала трансляции

На рисунке 23 представлена форма с информацией после завершения трансляции, которая состоит из следующих элементов: 1 – кнопка для закрытия формы, 2 – информация о продолжительности завершенной трансляции, 3 – информация о количестве просмотров, 4 – информация о игре (иконка игры и название игры).

Переход на данную форму будет происходить после завершения трансляции пользователем. При нажатии на кнопку закрытия форму будет осуществляться переход на главную форму приложения

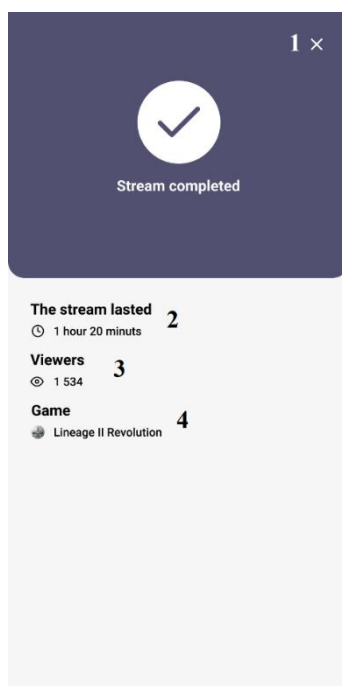


Рисунок 23 – Форма завершения трансляции

На рисунке 24 представлена форма с уведомлениями, состоящая из следующих компонентов: 1 – элемент списка, представляющий собой информацию о уведомлении. На элементе отображаются следующие данные: иконка канала, заголовок уведомления, время уведомления, название игры.

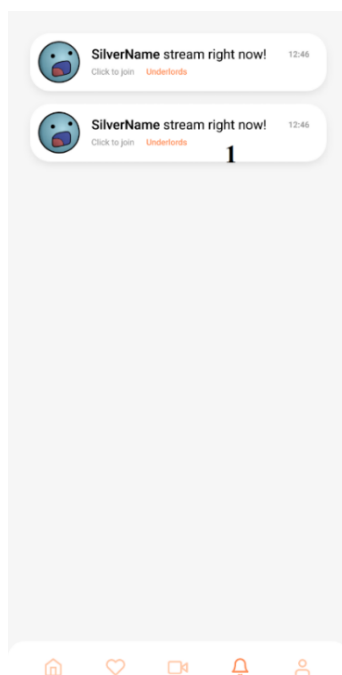


Рисунок 24 – Форма с уведомлениями

На рисунке 25 представлена форма профиля, которая состоит из следующих компонентов: 1 – кнопка для редактирования профиля, 2 – иконка профиля, 3 – текстовая информация о профиле (название канала и количество подписчиков), 4 – кнопка «Подписчики».

При нажатии на кнопку редактирования профиля будет открываться диалог для ввода и сохранения новых данных профиля. При нажатии на кнопку «Подписчики» будет осуществляться переход на форму со списком подписчиков.

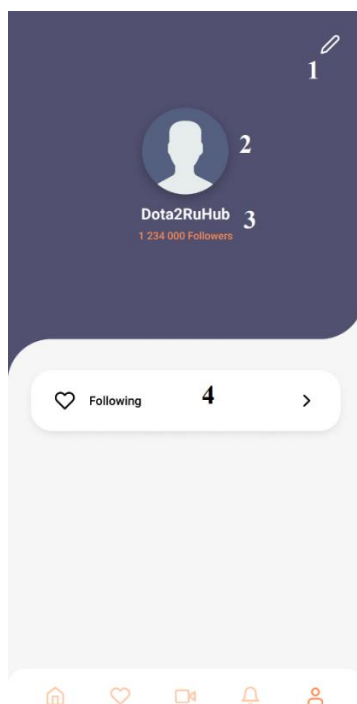


Рисунок 25 – Форма профиля

На рисунке 26 представлен диалог для редактирования профиля. На данном рисунке цифрами отмечены: 1 – кликабельная иконка профиля для изменения текущей иконки, 2 – текстовое поле ввода «Логин», 3 – текстовое поле ввода «Название канала», 4 – кнопка для сохранения изменений.

После нажатия кнопки сохранения диалог будет закрыт, а форма профиля обновлена.

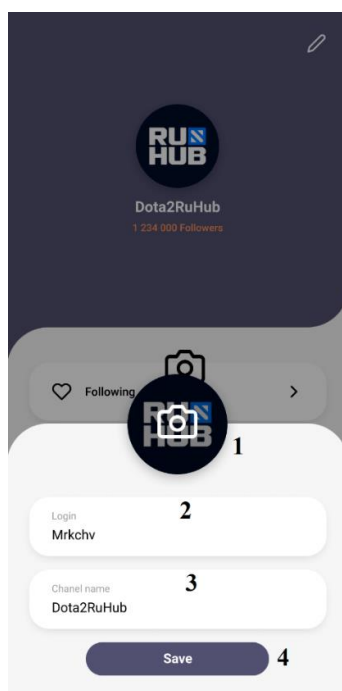


Рисунок 26 – Диалог редактирования профиля

На рисунке 26 представлен диалог для входа. На данном рисунке цифрами отмечены: 1 – текстовое поле ввода «Логин», 2 – текстовое поле ввода «Название канала», 3 – кнопка для входа.

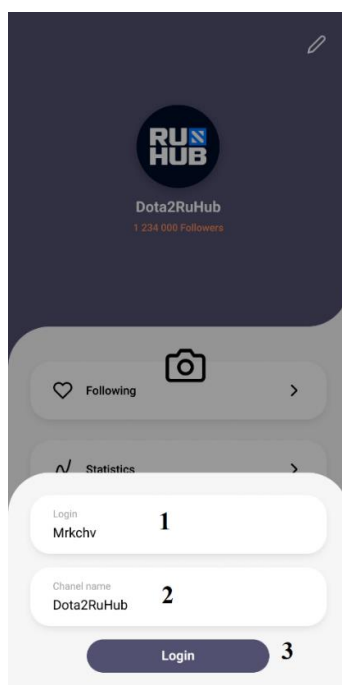


Рисунок 27 – Диалог входа в аккаунт

На рисунке 28 представлен диалог для регистрации в приложении. На данном рисунке цифрами следующие элементы: 1 – текстовое поле ввода «Логин», 2 – текстовое поле ввода «Название канала», 3 – текстовое поле ввода «Пароль», 4 – кнопка для регистрации.

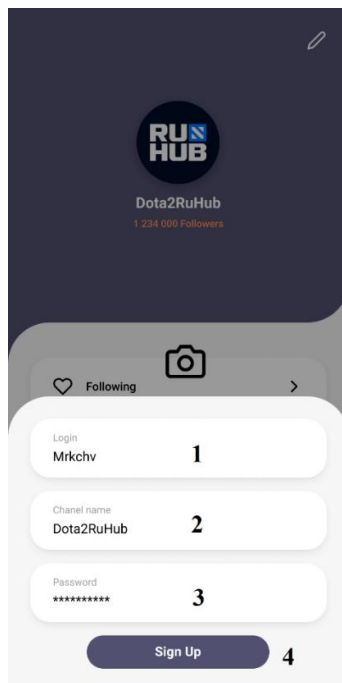


Рисунок 28 – Диалог регистрации в приложении

В результате, было проведено проектирование интерфейса мобильного приложения.

2.4 Проектирование серверной части приложения

2.4.1 Концептуальное и логическое проектирование базы данных

Предметной областью разработки является мобильное приложение для просмотра игрового контента с использованием потокового видео. Для предметной области была построена концептуальная модель, представленная на рисунке 29.

Концептуальная модель – это отражение предметной области, для которой разрабатывается база данных.

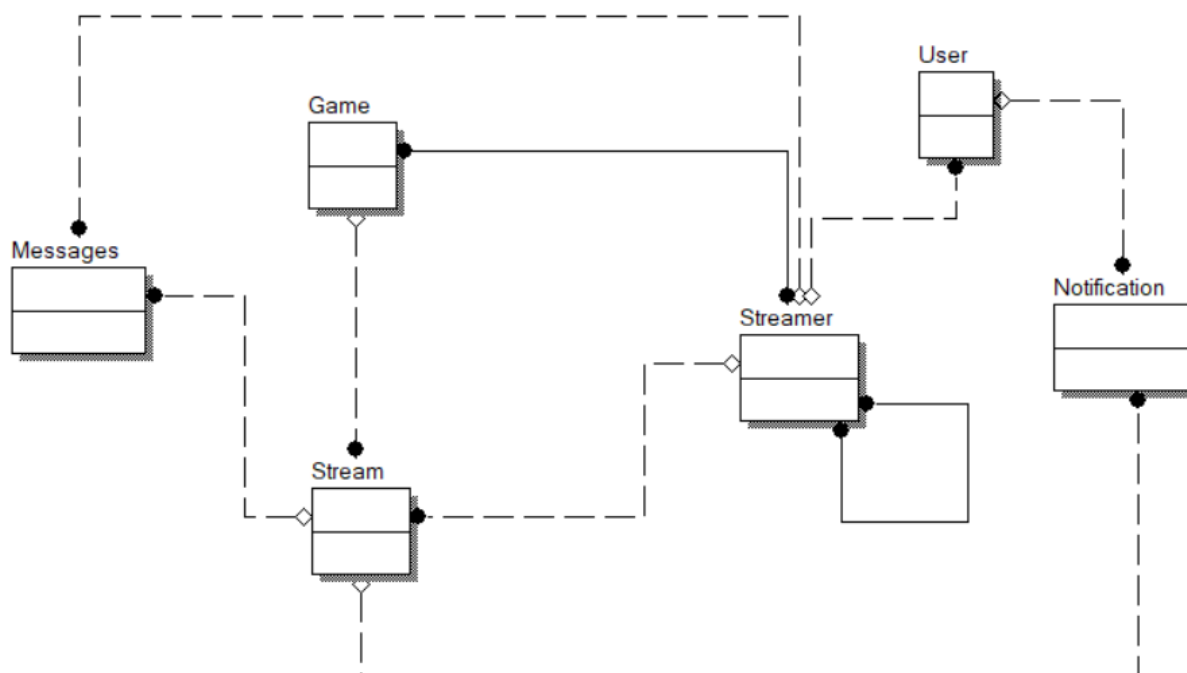


Рисунок 29 – Концептуальная модель предметной области

На основании концептуальной модели были выделены следующие типы сущностей (таблица 9).

Таблица 9 – Сущности предметной области

№	Название	Описание	Категория	Количество экземпляров	Режим обновления
1	User	Список пользователей	Стержневая	1000	25% в год
2	Streamer	Список стримеров	Характеристическая	1000	25% в год
3	Game	Список игр с описанием	Стержневая	100	40% в год
4	Stream	Список стримов	Стержневая	10000	60% в год
5	Messages	Список сообщений	Характеристическая	100000	
6	Notification	Список уведомлений	Характеристическая	10000	60% в год

Для сущностей предметной области были сформированы определенные связи, представленные в таблице 10.

Таблица 10 – Типы связей между сущностями предметной области

№	Тип сущности 1	Тип сущности 2	Мощность	Обязательность
1	User	Streamer	1:1	Обязательно
2	User	Notification	1:M	Обязательно
3	Streamer	Game	M:M	Необязательно (Стример может быть не подписан на игры)
4	Streamer	Streamer	M:M	Необязательно (Стример может быть не подписан на других стримеров)
5	Streamer	Messages	1:M	Обязательно
6	Stream	Messages	1:M	Обязательно
7	Stream	Game	M:1	Обязательно
8	Stream	Streamer	M:1	Обязательно
9	Stream	Notification	1:M	Обязательно

Логическая модель, построенная в нотации IDEF1X, представлена на рисунке 30.

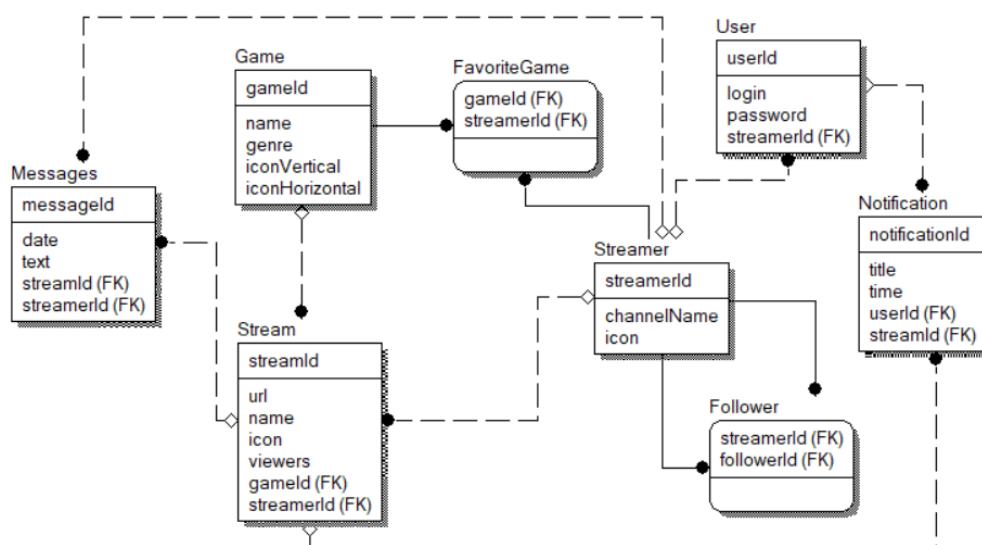


Рисунок 30 – Логическая модель базы данных

Логическая модель данных – это расширение концептуальной модели данных, которая включает в себя все сущности, атрибуты, ключи и взаимосвязи, которые представляют бизнес-информацию и определяют бизнес-правила.

2.5 Проектирование серверной части приложения

В качестве архитектурного подхода для серверной части был выбран REST API. Данный подход накладывает ограничения на разрабатываемое приложение [23]:

- модель клиент-сервер;
- отсутствие состояния (на сервере не хранится информация о состоянии клиента);
- кэширование (клиенты и промежуточные узлы могут кэшировать ответы сервера).

Клиент будет взаимодействовать с сервером посредством HTTP-запросов. Данные передаваемые между клиентом и сервером будут представлены в формате JavaScript Object Notation (JSON).

Серверная часть разработана на языке Java с использованием Spring Boot. Spring Boot – это библиотека, которая позволяет проще и быстрее разрабатывать приложения на основе Spring [24].

3 Реализация мобильного приложения

3.1 Реализация серверной части

3.1.1 Реализация базы данных

На основе логической модели предметной области была разработана физическая модель базы данных, представленная на рисунке 31. Моделирование проводилось в среде ErWin Data Modeler.

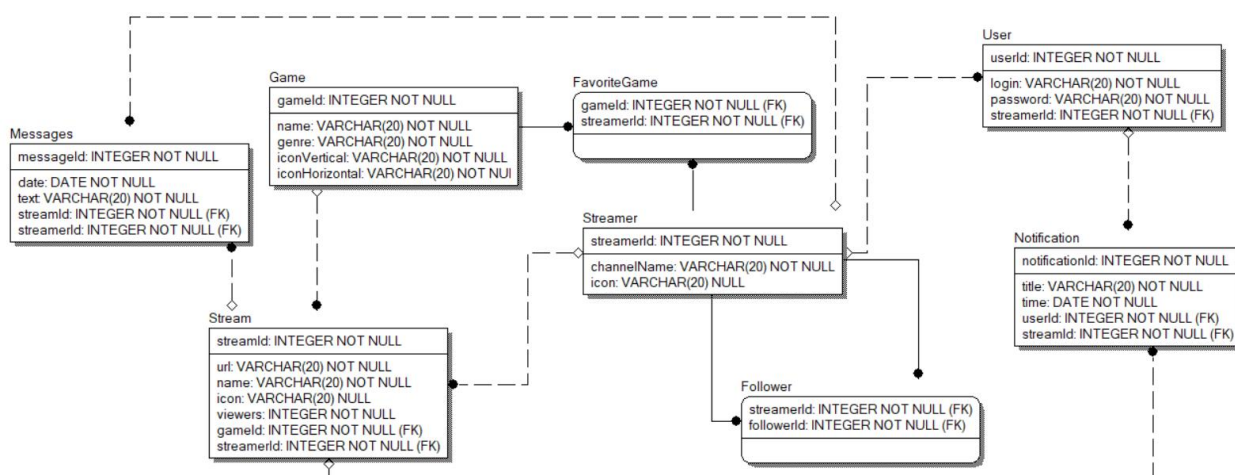


Рисунок 31 – Физическая модель базы данных

Создание базы данных и соответствующих таблиц было выполнено с помощью pgAdmin. Это платформа для администрирования и разработки баз данных [25].

База данных была развернута на локальном сервере PostgreSQL.

3.1.2 Реализация серверной части приложения

Разработка серверной части проводилась в среде программирования IntelliJ IDEA Community 2020.3 на языке программирования Java с использованием Spring Boot. Листинг серверной части представлен в приложении Б.

Библиотека Spring Boot была выбрана, т.к. позволяет избежать сложностей, возникающих при настройке Java-приложения на основе Spring.

Основными особенностями библиотеки является автоматическая конфигурация приложения и встроенная поддержка сервера приложений [26].

На рисунке 32 представлена диаграмма классов серверной части.

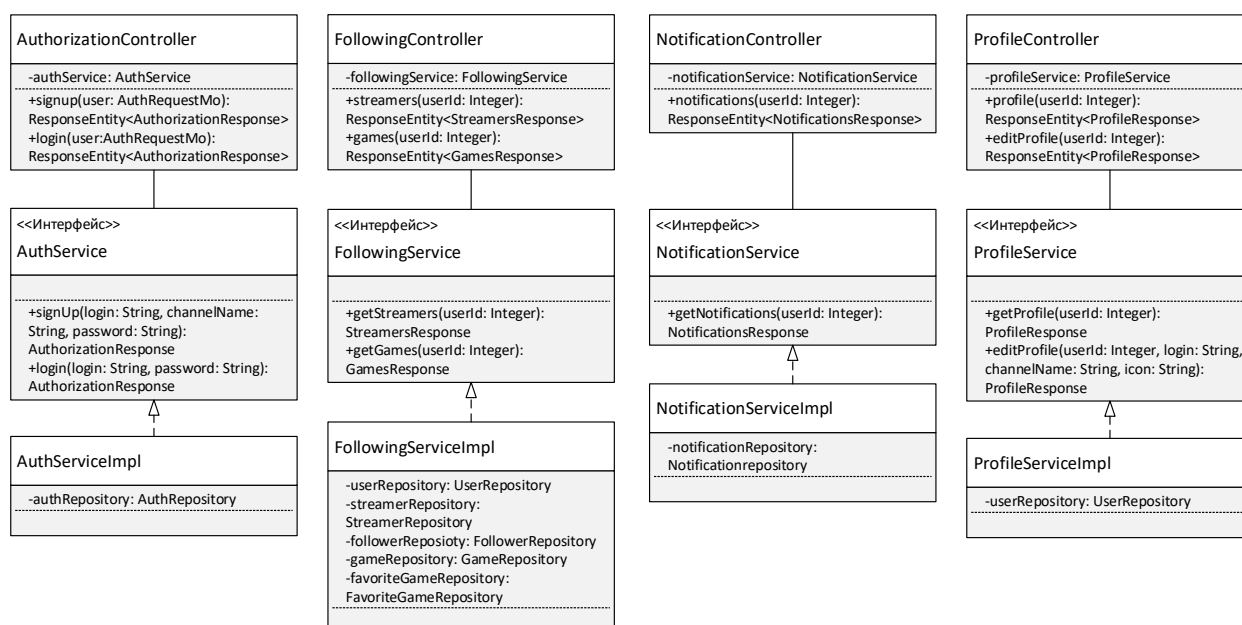


Рисунок 32 – Диаграмма классов серверной части

Архитектурно серверная часть разбивается на определенный вид классов, каждый из которых выполняет определенные действия. Выделяются такие классы как: Controller, Service и Repository.

Класс Controller (приложения Б.2-Б.6) отвечает за перехват входящих запросов, отправляет данные в классы Service, получает данные от классов Service и возвращает HTTP-ответ в формате JSON [27].

Класс Service (приложения Б.7-Б.11) содержат в себе бизнес-логику, а также преобразует классы данных, полученные из базы данных в классы, которые затем передаются в Controller.

Интерфейсы Repository реализуют интерфейс JpaRepository, который изначально имеет много функций для операций с базами данных [28], и как следствие нужны для получения/изменения/добавления записей в базу данных.

3.2 Реализация клиентской части мобильного приложения

Мобильное приложение разрабатывалось на языке программирования Kotlin в среде Android Studio. Листинг клиентской части представлен в приложении В.

Клиентская часть включает в себя классы, предназначенные для отображения данных, полученных от сервера, классы, реализующие бизнес-логику, а также классы необходимые для взаимодействия с сетью.

Разработанное приложение представляет собой многомодульное приложение. Были выделены следующие модули [29]:

- app (основной модуль приложения содержащий в себе класс приложения и главное Activity);
- core (модуль содержащий в себе классы для взаимодействия с сетью, а также классы, определяющие базовое поведение всех Activity и Fragment);
- core-views (модуль содержащий кастомные представления, а также классы для взаимодействия с представлениями, которые используются сразу в нескольких модулях);
- feature-home (модуль, содержащий классы, относящиеся к главной форме приложения и к форме трансляции);
- feature-following (модуль, содержащий классы, относящиеся к форме подписок);
- feature-stream (модуль, содержащий классы, относящиеся к форме начала трансляции);
- feature-notification (модуль, содержащий классы, относящиеся к форме уведомлений);
- feature-profile (модуль, содержащий классы, относящиеся к форме профиля).

На рисунке 33 представлены зависимости модулей приложения друг от друга.

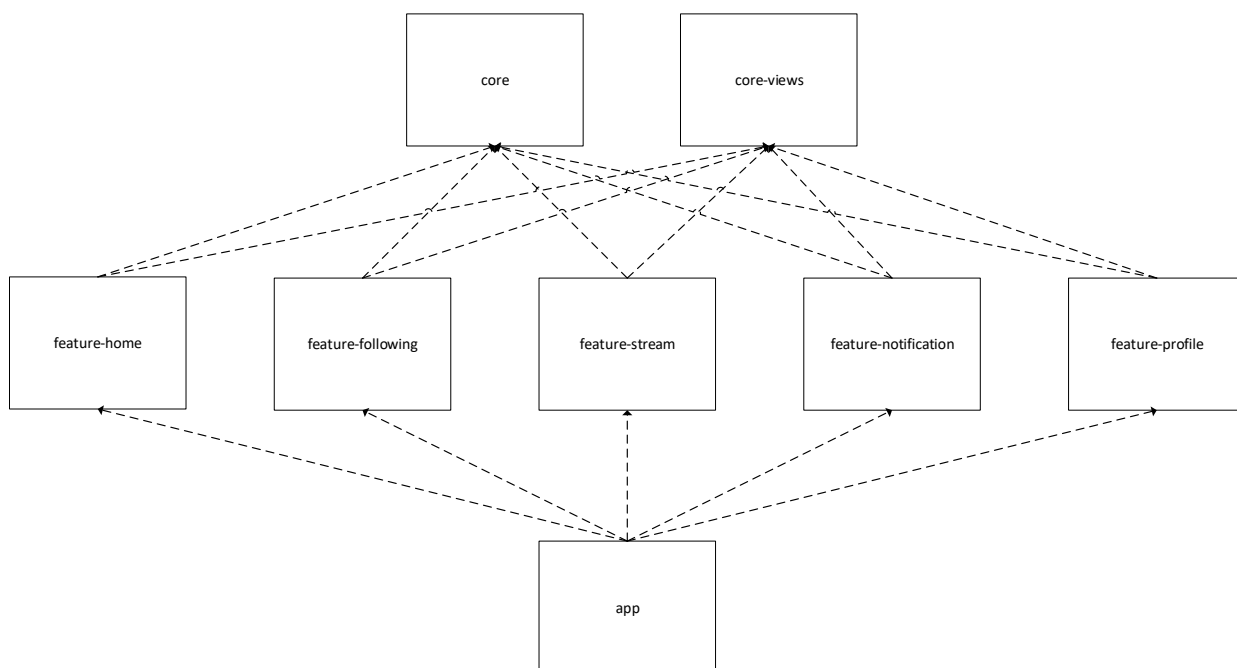


Рисунок 33 – Зависимости модулей приложения

Из приведенного рисунка видно, что `app` модуль имеет доступ ко всем остальным модулям приложения. Модули с префиксом `feature` находятся на одном уровне и не могут напрямую взаимодействовать друг с другом. Данные модули содержат в себе бизнес-логику приложения. Они имеют доступ только к `core` модулям. `Core` модули в свою очередь также не могут взаимодействовать друг с другом и содержат в себе основные классы для взаимодействия с сетью, базой данных (если есть), а также классы, которые являются общими для `feature` модулей [30].

На рисунке 34 приведена диаграмма компонентов, отражающая общую структуру исходного кода программы. Компонентами данного приложения являются файлы, содержащие исходный код фрагментов, в которых происходит инициализация экранов, и `ViewModel`, являющиеся частью паттерна `MVVM`. Данные фрагменты используются в классе `MainActivity` (приложение В.21), в котором происходит управление навигацией приложения.

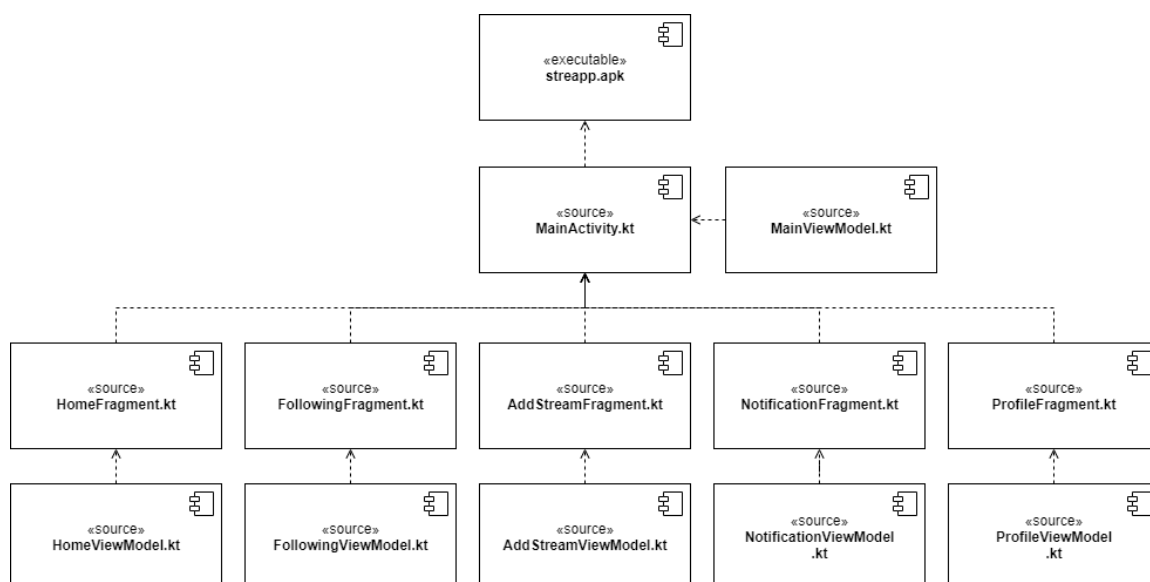


Рисунок 34 – Диаграмма компонентов мобильного приложения

На рисунке 35 представлена диаграмма развертывания, отображающая физическое расположение компонентов системы.

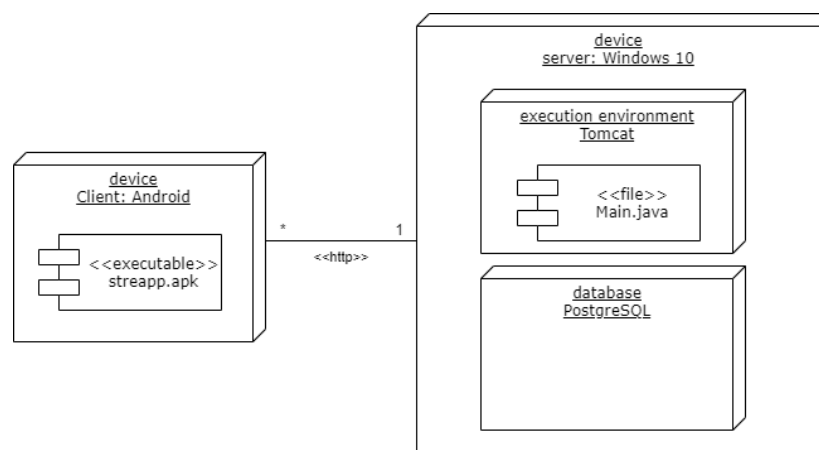


Рисунок 35 – Диаграмма развертывания мобильного приложения

Диаграмма отражает взаимодействие клиентов с помощью протокола HTTP. Клиентская часть работает на мобильных устройствах под управлением ОС Android, а серверная часть – на компьютере под управлением ОС Windows и выполняется в среде Tomcat. Также на серверной части располагается база данных.

4 Тестирование программного продукта

4.1 Тестирование серверной части приложения

Тестирование разработанного веб-сервиса осуществлялось с помощью инструментального средства Postman.

В таблице 11 представлены результаты тестирования всех функциональных возможностей веб-сервиса.

Таблица 11 – Результаты тестирования веб-сервиса

№	Описание теста	Запрос		Результат
		Метод	Параметры	
1	Регистрация	POST	Login=Mrkchv, ChannelName = Lazurit Password=1233	Тест пройден. Пользователь добавлен. Код ответа HTTP – 200 (рисунок Г.1)
2	Вход	POST	Login=Mrkchv Password=1233	Тест пройден. Сервер возвращает идентификатор пользователя. Код ответа HTTP – 200 (рисунок Г.2)
3	Получение списка подписок	GET		Тест пройден. Сервер возвращает список подписок или ошибку при пустом списке. Код ответа HTTP – 200 (рисунок Г.3)

Продолжение таблицы 11

№	Описание теста	Запрос		Результат
		Метод	Параметры	
4	Получение списка избранных игр	GET		Тест пройден. Сервер возвращает список избранных игр или ошибку при пустом списке. Код ответа HTTP – 200 (рисунок Г.4)
5	Получение списка уведомлений	GET		Тест пройден. Сервер возвращает список уведомлений или ошибку при пустом списке. Код ответа HTTP – 200 (рисунок Г.5)
6	Получение данных профиля	GET		Тест пройден. Сервер возвращает данные профиля или ошибку если пользователь не авторизован. Код ответа HTTP – 200 (рисунок Г.6)

Окончание таблицы 11

№	Описание теста	Запрос		Результат
		Метод	Параметры	
7	Получение списка подписчиков	GET		Тест пройден. Сервер возвращает список подписчиков. Код ответа HTTP – 200 (рисунок Г.7)
8	Получение данных трансляции	GET	StreamId=1	Тест пройден. Сервер возвращает данные трансляции. Код ответа HTTP – 200 (рисунок Г.8)
9	Начало трансляции	POST	StreamName=My stream Game=Underlors	Тест пройден. Сервер добавляет данные трансляции. Код ответа HTTP – 200 (рисунок Г.9)

Результаты тестов показывают корректность работы веб-сервиса.

4.2 Тестирование клиентской части приложения

4.2.1 Функциональное тестирование

Для проверки корректности работы клиентской части приложения было проведено функциональное тестирование с помощью Android-устройства Honor Play. Результаты тестов представлены в таблице 12.

Таблица 12 – Результаты функционального тестирования

№ п/п	Вариант использования	Описание теста	Результаты теста
1	Зарегистрироваться в приложении	Перейти на форму «Профиль». Нажать на кнопку «Зарегистрироваться». В появившемся диалоге заполнить данные и нажать «Зарегистрироваться»	Тест выполнен. Отображается форма «Профиль» (рисунок Д.1, рисунок Д.2).
2	Выполнить вход в приложении	Перейти на форму «Профиль». Нажать на кнопку «Войти». В появившемся диалоге заполнить данные и нажать «Войти»	Тест выполнен. Отображается форма «Профиль» (рисунок Д.3, рисунок Д.4).
3	Просмотреть список трансляций	Перейти на форму «Главная»	Тест выполнен. Список трансляций отображается на главной (рисунок Д.5).
4	Выбрать трансляцию	Выбрать трансляцию из списка на главной	Тест выполнен. Отображается форма «Трансляция» (рисунок Д.6).
5	Написать сообщение в чат	На форме «Трансляция» нажать на кнопку «Сообщение». В появившемся диалоге ввести текст сообщения и нажать кнопку «Отправить»	Тест выполнен. Сообщение добавлено (рисунок Д.7).
6	Подписаться на другого пользователя	На форме «Трансляция» или «Подписчики» выбрать другого пользователя и нажать «Подписаться»	Тест выполнен. Другой пользователь добавлен в подписки (рисунок Д.8, рисунок Д.9).
7	Отписаться от пользователя	На форме «Трансляция», «Подписки» или «Подписчики» выбрать пользователя и нажать «Отписаться»	Тест выполнен. Другой пользователь удален из подписок (рисунок Д.10, рисунок Д.11).
8	Подписаться на игру	Выбрать игру. В появившемся диалоге нажать «Подписаться»	Тест выполнен. На диалоге с информацией о игре есть кнопка «Отписаться» (рисунок Д.12).
9	Отписаться от игры	Выбрать игру из списка. В появившемся диалоге нажать «Отписаться»	Тест выполнен. На диалоге с информацией о игре есть кнопка «Подписаться» (рисунок Д.13).
10	Начать собственную трансляцию	На форме «Начать трансляцию» заполнить поля и выбрать игру.	Тест выполнен. Запущена игра (рисунок Д.14).

Продолжение таблицы 12

№ п/п	Вариант использования	Описание теста	Результаты теста
11	Закончить трансляцию	Во время записи игрового процесса нажать на кнопку «Завершить»	Тест выполнен. Отображается форма «Информация о трансляции» (рисунок Д.15).
12	Посмотреть профиль	Перейти на форму «Профиль»	Тест выполнен. Открыта форма «Профиль» с информацией (рисунок Д.16).
13	Посмотреть список подписчиков	Перейти на форму «Профиль» и нажать на кнопку «Подписчики»	Тест выполнен. Открыта форма «Подписчики» (рисунок Д.8).
14	Редактировать данные профиля	На форме «Профиль» нажать кнопку для редактирования профиля. В появившемся диалоге ввести новые данные профиля и нажать кнопку «Сохранить»	Тест выполнен. Открыта форма «Профиль» с обновленными данными (рисунок Д.16, рисунок Д.17).
15	Посмотреть список подписок	Перейти на форму «Подписки»	Тест выполнен. Открыта форма «Подписки» и отображается список подписок (рисунок Д.18).
16	Посмотреть уведомления	Перейти на форму «Уведомления»	Тест выполнен. Отображается форма «Уведомления» и список уведомлений (рисунок Д.19).

Функциональное тестирование показало корректность работы мобильного приложения.

4.3 Метрики кода

Метрикой программного кода называется количественная мера, позволяющая определить, в какой степени система, компоненты или процесс, обладают заданным атрибутом. Все метрики программного кода можно разделить на группы:

- метрики размера программы (SLOC-метрики);
- метрики сложности потока управления программ (метрики

Холстеда и цикломатическая сложность).

В листинге 1 приведен класс `FollowingViewModel`. Поля данного класса отражают элементы `View`. Поле `repository` используется для получения данных.

Листинг 1 – Класс `FollowingViewModel`

```
class FollowingViewModel(private val repository: FollowingRepository)
: BaseViewModel() {

    val streamers = DisplayableData<List<Streamer>>>()
    val favoriteGames = DisplayableData<List<Game>>>()
    val showGameInfo = Action<GameItem>()

    fun onCreateView() {
        getGames()
        getStreamers()
    }

    fun getStreamers() {
        launch(
            block = { repository.getStreamers() },
            loading = { onStreamersLoading(it) },
            successHandler = { onStreamersGet(it) }
        )
    }

    fun onStreamersLoading(isLoading: Boolean) {
        if (isLoading) streamers.show(
            listOf(
                StreamerLoader,
                StreamerLoader,
                StreamerLoader,
                StreamerLoader
            )
        )
    }

    fun onStreamersGet(streamers: StreamersResponseMo) {
        this.streamers.show(streamers.items.toStreamers())
    }

    fun getGames() {
        launch(
            block = { repository.getFavoriteGames() },
            loading = { onFavoriteGamesLoading(it) },
            successHandler = { onFavoriteGamesGet(it) }
        )
    }

    fun onFavoriteGamesLoading(isLoading: Boolean) {
        if (isLoading) favoriteGames.show(listOf(GameLoader,
            GameLoader, GameLoader))
    }
}
```

```

fun onFavoriteGamesGet(games: GamesResponseMo) {
    favoriteGames.show(games.items.toGames())
}

fun onGameClick(game: GameItem) {
    showGameInfo.call(game)
}
}

```

На примере данного класса будут рассчитаны метрики кода.

4.3.1 SLOC-метрика

SLOC-метрика предназначена для подсчета физических и логических строк программного кода, к физическим строкам относят все непустые строки программного файла, к логическим – число строк с командами, использующимися в программном коде. Так же SLOC-метрика предполагает вычисление строк с комментариями.

Вычислим SLOC-метрику для приведенного класса. Результаты вычисления приведены в таблице 13.

Таблица 13 – Вычисление SLOC-метрики

Вид строк	Количество
Физические строки	75
Логические строки	34
Строки-комментарии	0

Количество физических строк кода может различаться для различных программистов, т.к. зачастую каждый программист может по-своему оформить программный код. А количество логических строк кода будет различным только для программного кода, написанного на различных языках программирования.

4.3.2 Метрики Холстеда

Метрики Холстеда включают в себя следующие измеряемые характеристики программы:

- количество уникальных операторов программы;

- количество уникальных операндов программы;
- общее количество операторов программы;
- общее количество операндов программы.

На основе данных метрик можно получить следующие количественные оценки:

- словарь программы равный числу уникальных операторов плюс число уникальных операндов;
- длина программы равная общему числу операторов плюс общее число операндов;
- объем программы равный длине программы, умноженной на $\log_2$ словаря программы.

Также используя метрики, описанные выше можно рассчитать комплексный показатель качества кода по формуле:

$$M = \max\left(0, (171 - 5.2 * \ln(V) - 0.23 * CC - 16.2 * \ln(LoC)) * \frac{100}{171}\right),$$

где V – объём программы;

CC – структурная сложность кода;

LoC – количество строк кода.

Для приведенного класса получим метрики Холстеда. В таблицах 14 и 15 отображены уникальные операторы программного кода и уникальные операнды.

Таблица 14 – Уникальные операторы

Номер	Оператор	Количество
1	class	1
2	private	1
3	val	4
4	<>	5
5	()	4
6	fun	8
7	=	3
8	this	1
9	:	7
10	if	2

Таблица 15 – Уникальные операнды

Номер	Операнд	Количество
1	repository	4
2	streamers	3
3	favoriteGames	3
4	showGameInfo	2

Из таблиц 8 и 9 видно, что общее число операторов программного кода равно 36, а общее число операндов равно 12.

Словарь программы $10 + 4 = 14$.

Длина программы $36 + 12 = 48$.

Объем программы $48 * \log_2 12 = 51.79$.

Количество строк кода LoC = 56.

Структурная сложность кода CC = 3.

Комплексный показатель качества кода M = 49.458.

Комплексный показатель качества кода принимает значения от 0 до 100. Чем больше значение этого показателя, тем проще поддерживать код. Полученное значение составляет 49.458 и означает среднюю сложность поддержки кода.

Заключение

В рамках выполнения выпускной квалификационной работы было разработано мобильное приложение для организации просмотра и записи игрового контента с использованием потокового видео.

На этапе анализа требований был проведен сравнительный анализ мобильных приложений-аналогов, определена целевая аудитория, выделены функциональные и нефункциональные требования к разрабатываемому приложению.

Было осуществлено планирование разработки мобильного приложения, оценена стоимость такой разработки. Планирование показало, что срок реализации проекта составит 68 дней, а стоимость составит 198 140 рублей. Полученные результаты свидетельствуют об экономической целесообразности данного проекта.

На этапе проектирования приложения были составлены концептуальная и логическая модели базы данных, определена архитектура разрабатываемого приложения.

Мобильное приложение было разработано с использованием языка Kotlin в среде разработки Android Studio. Серверная часть была разработана с использованием языка Java и библиотеки Spring Boot в среде разработки IntelliJ IDEA.

Было проведено тестирование серверной и клиентской части разработанного приложения, которое показало корректность работы всего приложения.

В дальнейшем планируется развертывание серверной части в управляемой среде Google Kubernetes Engine и размещение разработанного мобильного приложения в магазине приложений Google Play.

Таким образом, поставленная задача по реализации мобильного приложения для организации записи и просмотра игрового контента с использованием потокового видео была выполнена в полном объеме.

Список использованных источников

1. Программа «Цифровая экономика Российской Федерации». Утверждена распоряжением Правительства Российской Федерации от 28 июля 2017 г. № 1632-р. [Электронный ресурс]. URL: <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf> (дата обращения: 23.03.2021)
2. Распоряжение Президента Российской Федерации от 9 мая 2017 г. № 203 «Стратегия развития информационного общества в Российской Федерации на 2017 – 2030 годы» [Электронный ресурс]. URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 23.03.2021)
3. Пишем свой протокол поверх UDP / Протоколы передачи видео [Электронный ресурс]. URL: <http://habr.com/ru/company/oleg-bunin/blog/413479/> (дата обращения: 23.03.2021)
4. Эти сервисы нужны каждому стримеру. [Электронный ресурс]. URL: <http://blog-themafia-name.turbopages.org/blog.themafia.name/s/eti-siervisy-nuzhny-kazhdomu-strimieru> (дата обращения: 23.03.2021)
5. Рынок игрового стриминга в России и СНГ превысил 20 млрд руб. [Электронный ресурс]. URL: https://www.rbc.ru/technology_and_media/24/12/2019/5e01d1659a7947d1b61e5e3e (дата обращения: 25.04.2021)
6. Twitch занял 91% рынка стриминга во всём мире по количеству транслированных часов [Электронный ресурс]. URL: <https://vc.ru/services/165025-twitch-zanyal-91-rynka-striminga-vo-vsem-mire-po-kolichestvu-translirovannyh-chasov> (дата обращения: 25.04.2021)
7. Рейтинг мобильных процессоров конец 2018 – начало 2019 года [Электронный ресурс]. URL: <https://stevsky.ru/mobilnie-protssessori/mobilnie-protssessori-2019-bolshoy-obzor-vsech-mobilnich-soc-predstavlennich-na-rinke-v-2019-godu> (дата обращения: 26.04.2021)

8. Рейтинг мобильных процессоров для смартфонов – апрель 2021 [Электронный ресурс]. URL: <https://nanoreview.net/ru/soc-list/rating> (дата обращения: 26.04.2021)
9. Omlet Arcade – запись экрана и стрим мобильных игр [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=mobisocial.arcade&hl=ru&gl=US> (дата обращения: 27.04.2021)
10. Streamlabs: Live Streaming App [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.streamlabs&hl=ru&gl=US> (дата обращения: 27.04.2021)
11. Twitch [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=tv.twitch.android.app&hl=ru&gl=US> (дата обращения: 27.04.2021)
12. Mobile Operating System Market Share Worldwide [Электронный ресурс]. URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (дата обращения: 28.04.2021)
13. Смартфоны (рынок России) 2021 [Электронный ресурс]. URL: [https://www.tadviser.ru/index.php/Статья:Смартфоны_\(рынок_России\)](https://www.tadviser.ru/index.php/Статья:Смартфоны_(рынок_России)) (дата обращения: 28.04.2021)
14. История версий Android [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/История_версий_Android (дата обращения: 28.04.2021)
15. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование, 2е издание / Д. Арлоу, И. Нейштадт – СПб: Символ Плюс, 2007. – 624 с.
16. Статья на Wikipedia о UML [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/UML> (дата обращения: 30.04.2021)
17. Боронина Л. Н. Основы управления проектами / Л. Н. Боронина, З. В. Сенук – Екатеринбург: изд-во Урал. ун-та, 2015. — 112 с.

18. Управление проектами и рисками // Томский Политехнический Университет [Электронный ресурс]. URL: https://portal.tpu.ru/SHARED/s/SIRAZITDINOVA/biography/course_materials/Tab1/L8.pdf (дата обращения: 30.04.2021)
19. Джонсон Тимоти. Microsoft Project 2016. Шаг за шагом / Джонсон Тимоти, Четфилд Карл – М.: КТК Галактика, 2018. – 594 с.
20. НОУ ИНТУИТ. Лекция. Введение в технологии разработки программного обеспечения [Электронный ресурс]. URL: [https://intuit.ru/studies/courses/4806/1054/lecture/16117#:~:text=Технология%20разработки%20программного%20обеспечения\(ПО\)%20представляет,средства%20и%20процедуры%20разработки%20ПО](https://intuit.ru/studies/courses/4806/1054/lecture/16117#:~:text=Технология%20разработки%20программного%20обеспечения(ПО)%20представляет,средства%20и%20процедуры%20разработки%20ПО) (дата обращения: 02.05.2021)
21. Гради Буч. Язык UML. Руководство пользователя / Гради Буч, Джеймс Рамбо, Ивар Якобсон. – М.: ДМК Пресс, 2007. – 496 с.
22. Metanit. Паттерн MVVM [Электронный ресурс]. URL: <https://metanit.com/sharp/wpf/22.1.php> (дата обращения: 02.05.2021)
23. REST // Википедия [Электронный ресурс]. URL: <https://ru.wikipedia.org/wiki/REST> (дата обращения: 02.05.2021)
24. Spring Boot [Электронный ресурс]. URL: <https://spring.io/projects/spring-boot> (дата обращения: 02.05.2021)
25. pgAdmin. PostgreSQL Tools [Электронный ресурс]. URL: <https://www.pgadmin.org/> (дата обращения: 25.05.2021)
26. Введение в Spring Boot: создание простого REST API на Java [Электронный ресурс]. URL: <https://habr.com/ru/post/435144/> (дата обращения: 26.05.2021)
27. Quick guide to Spring Controllers [Электронный ресурс]. URL: <https://www.baeldung.com/spring-controllers> (дата обращения: 26.05.2021)
28. Spring Boot – Accessing Data with JPA Hibernate and PostgreSQL [Электронный ресурс]. URL: <https://medium.com/analytics-vidhya/spring-boot-accessing-data-with-jpa-hibernate-and-postgresql-af68386363a4> (дата обращения: 26.05.2021)

29. Modularizing Android Application [Электронный ресурс]. URL: <https://medium.com/google-developer-experts/modularizing-android-applications-9e2d18f244a0> (дата обращения: 27.05.2021)

30. Советы по модуляризации приложений Android [Электронный ресурс]. URL: <https://zen.yandex.ru/media/nuancesprog/sovety-po-moduliarizacii-prilozhenii-android-60379298dfc3fd7dc21147f2> (дата обращения: 27.05.2021)

Приложение А
ГЛОССАРИЙ
(справочное)

Стриминг (англ. streaming) – онлайн вещание, которое ведет пользователь, без перерывов.

MVVM (Model-View-ViewModel) – архитектурный паттерн, позволяющий отделить логику приложения от визуальной части.

REST (Representational State Transfer) – архитектурный стиль взаимодействия компонентов распределенного приложения в сети.

API (Application Programming Interface) – набор готовых классов, функций, процедур, констант и структур, предоставляемых приложением или операционной системой для использования во внешних программных продуктах.

RUP (Rational Unified Process) – методология разработки программного обеспечения, которая использует итеративную модель разработки.

SQL (Structured Query Language) – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, основанный на технологии «клиент-сервер».

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

SLOC (Source Lines of Code) – это метрика программного обеспечения, используемая для измерения его объема с помощью подсчета количество строк в тексте исходного кода.

Приложение Б
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

Листинг Б.1 – Файл Main.java

```
@SpringBootApplication
public class Main {

    public static void main(String[] args) {
        SpringApplication.run(Main.class, args);
    }

}
```

Листинг Б.2 – Файл AuthorizationController.java

```
@RestController
public class AuthorizationController {

    private final AuthService authService;

    @Autowired
    public AuthorizationController(AuthService authService) {
        this.authService = authService;
    }

    @PostMapping(value = "/signup")
    public ResponseEntity<AuthorizationResponse>
    signup(@RequestBody SignUpRequestMo user) {
        AuthorizationResponse authorizationResponse =
        authService.signup(user.getLogin(), user.getChannelName(),
        user.getPassword());
        return new ResponseEntity<>(authorizationResponse,
        HttpStatus.OK);
    }

    @PostMapping(value = "/login")
    public ResponseEntity<AuthorizationResponse>
    login(@RequestBody LoginRequestMo user) {
        AuthorizationResponse authorizationResponse =
        authService.login(user.getLogin(), user.getPassword());
        return new ResponseEntity<>(authorizationResponse,
        HttpStatus.OK);
    }

}
```

Листинг Б.3 – Файл FollowingController.java

```
@RestController
public class FollowingController {

    private final FollowingService followingService;

    @Autowired
    public FollowingController(FollowingService
    followingService) {
```

```

        this.followingService = followingService;
    }

    @GetMapping(value = "/streamers/following")
    public ResponseEntity<StreamersResponse>
    streamers(@RequestHeader("USER_ID") Integer userId) {
        return new
    ResponseEntity<>(followingService.getStreamers(userId),
    HttpStatus.OK);
    }

    @GetMapping(value = "/games")
    public ResponseEntity<GamesResponse>
    games(@RequestHeader("USER_ID") Integer userId) {
        return new
    ResponseEntity<>(followingService.getGames(userId),
    HttpStatus.OK);
    }
}

```

Листинг Б.4 – Файл NotificationController.java

```

@RestController
public class NotificationController {

    private final NotificationService notificationService;

    @Autowired
    public NotificationController(NotificationService
    notificationService) {
        this.notificationService = notificationService;
    }

    @GetMapping(value = "/notifications")
    public ResponseEntity<NotificationsResponse>
    notifications(@RequestHeader("USER_ID") Integer userId) {
        return new
    ResponseEntity<>(notificationService.getNotifications(userId),
    HttpStatus.OK);
    }
}

```

Листинг Б.5 – Файл ProfileController.java

```

@RestController
public class ProfileController {

    private final ProfileService profileService;

    @Autowired
    public ProfileController(ProfileService profileService) {
        this.profileService = profileService;
    }
}

```

```

    }

    @GetMapping(value = "/profile")
    public ResponseEntity<ProfileResponse>
    profile(@RequestHeader("USER_ID") Integer userId) {
        ProfileResponse profileResponse =
        profileService.getProfile(userId);
        return new ResponseEntity<>(profileResponse,
        HttpStatus.OK);
    }

    @PostMapping(value = "/profile/edit")
    public ResponseEntity<BaseResponse>
    editProfile(@RequestHeader("USER_ID") Integer userId,
    @RequestBody EditRequestMo user) {
        BaseResponse response =
        profileService.editProfile(userId, user.getLogin(),
        user.getChannelName());
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @GetMapping(value = "/streamers/followers")
    public ResponseEntity<StreamersResponse>
    followers(@RequestHeader("USER_ID") Integer userId) {
        StreamersResponse response =
        profileService.getFollowers(userId);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }
}

```

Листинг Б.6 – Файл StreamController.java

```

@RestController
public class StreamController {

    private final StreamService streamService;

    @Autowired
    public StreamController(StreamService profileService) {
        this.streamService = profileService;
    }

    @GetMapping(value = "/stream")
    public ResponseEntity<StreamResponse>
    stream(@RequestHeader("USER_ID") Integer userId,
    @RequestParam("streamId") Integer streamId) {
        StreamResponse response =
        streamService.getStream(userId, streamId);
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    @GetMapping(value = "/stream/follow")

```

```

        public ResponseEntity<BaseResponse>
follow(@RequestHeader("USER_ID") Integer userId,
@RequestParam("streamId") Integer streamId,
@RequestParam("isFollowing") Boolean isFollowing) {
            BaseResponse response = streamService.follow(streamId,
isFollowing);
            return new ResponseEntity<>(response, HttpStatus.OK);
        }

        @PostMapping(value = "/stream/add")
        public ResponseEntity<BaseResponse>
addStream(@RequestHeader("USER_ID") Integer userId, @RequestBody
StreamRequestMo stream) {
            BaseResponse response = streamService.addStream(userId,
stream.getStreamName(), stream.getGameName());
            return new ResponseEntity<>(response, HttpStatus.OK);
        }
    }
}

```

Листинг Б.7 – Файл AuthServiceImpl.java

```

@Service
public class AuthServiceImpl implements AuthService {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private StreamerRepository streamerRepository;

    @Override
    public AuthorizationResponse signUp(String login, String
channelName, String password) {
        Users user = new Users();
        user.setLogin(login);
        AuthorizationResponse userInfoResponse = new
AuthorizationResponse();

        Optional<Users> userFromDB =
userRepository.findOne(Example.of(user));
        if (userFromDB.isPresent()) {
            if (userFromDB.get().getLogin().equals(login)) {
                userInfoResponse.setErrorResponse(new
ErrorResponse(
                    Constants.BUSINESS_ERROR_CODE,
Constants.BusinessSubCode.USER_EXIST_SUB_CODE,
                    "Логин занят"
                ));
            }
            return userInfoResponse;
        }
    }
}

```

```

        Streamer streamer = new Streamer();
        streamer.setChannelName(channelName);
        user.setStreamer(streamer);

        userFromDB = userRepository.findOne(Example.of(user));
        if (userFromDB.isPresent()) {
            if
(userFromDB.get().getStreamer().getChannelName().equals(channelName)) {
                userInfoResponse.setErrorResponse(new
ErrorResponse(
                    Constants.BUSINESS_ERROR_CODE,
Constants.BusinessSubCode.USER_EXIST_SUB_CODE,
                    "Название канала занято"
                ));
            }
            return userInfoResponse;
        }

        user.setPassword(password);
        streamerRepository.save(streamer);

        userInfoResponse.setUserId(userRepository.save(user).getUserId());
    };

    return userInfoResponse;
}

@Override
public AuthorizationResponse login(String login, String
password) {
    Users user = new Users();
    user.setLogin(login);
    user.setPassword(password);
    AuthorizationResponse userInfoResponse = new
AuthorizationResponse();

    Optional<Users> userFromDB =
userRepository.findOne(Example.of(user));
    if (!userFromDB.isPresent()) {
        userInfoResponse.setErrorResponse(new ErrorResponse(
            Constants.BUSINESS_ERROR_CODE,
Constants.BusinessSubCode.USER_NOT_EXIST_SUB_CODE,
            "Неверный логин или пароль"
        ));
        return userInfoResponse;
    } else {
        userInfoResponse.setUserId(userFromDB.get().getUserId());
    }
}

```

```

        return userInfoResponse;
    }
}

```

Листинг Б.8 – Файл FollowingServiceImpl.java

```

@Service
public class FollowingServiceImpl implements FollowingService {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private StreamerRepository streamerRepository;
    @Autowired
    private FollowerRepository followerRepository;
    @Autowired
    private GameRepository gameRepository;
    @Autowired
    private FavoriteGameRepository favoriteGameRepository;

    @Override
    public StreamersResponse getStreamers(Integer userId) {
        Integer streamerId =
userRepository.findStreamerIdByUserId(userId);
        List<Follower> followers =
followerRepository.findFollowingByStreamerId(streamerId);

        StreamersResponse response = new StreamersResponse();
        List<StreamerResponse> streamers = new ArrayList<>();
        for (Follower follower : followers) {
            Optional<Streamer> streamer =
streamerRepository.findById(follower.getFollowerId().getFollower
Id());
            streamer.ifPresent(value -> streamers.add(
                DataModelToResponseModel.toStreamer(value)
            ));
        }

        if (streamers.isEmpty()) {
            response.setErrorResponse(new ErrorResponse(
                Constants.BUSINESS_ERROR_CODE,
                Constants.BusinessSubCode.EMPTY_LIST,
                "У вас нет подписок"
            ));
        } else {
            response.setItems(streamers);
        }

        return response;
    }
}

```

```

@Override
public GamesResponse getGames(Integer userId) {
    Integer streamerId =
userRepository.findStreamerIdByUserId(userId);
    List<FavoriteGame> favoriteGames =
favoriteGameRepository.findAllByStreamerId(streamerId);

    GamesResponse response = new GamesResponse();
    List<GameResponse> games = new ArrayList<>();
    for (FavoriteGame favoriteGame : favoriteGames) {
        Optional<Game> game =
gameRepository.findById(favoriteGame.getFavoriteGameId().getGame
Id());
        game.ifPresent(value -> games.add(
            DataModelToResponseModel.toGame(value)
        ));
    }

    if (games.isEmpty()) {
        response.setErrorResponse(new ErrorResponse(
            Constants.BUSINESS_ERROR_CODE,
            Constants.BusinessSubCode.EMPTY_LIST,
            "У вас пока нет избранных игр"
        ));
    } else {
        response.setItems(games);
    }

    return response;
}
}

```

Листинг Б.9 – Файл NotificationServiceImpl.java

```

@Service
public class NotificationsServiceImpl implements
NotificationService {

    @Autowired
    private NotificationRepository notificationRepository;

    @Override
    public NotificationsResponse getNotifications(Integer
userId) {
        List<Notification> notifications =
notificationRepository.findAllByUserId(userId);

        NotificationsResponse response = new
NotificationsResponse();

        if (notifications.isEmpty()) {
            response.setErrorResponse(new ErrorResponse(

```

```

        Constants.BUSINESS_ERROR_CODE,
        Constants.BusinessSubCode.EMPTY_LIST,
        "Список уведомлений пуст"
    ));
    } else {

response.setNotifications(DataModelToResponseModel.toNotificatio
ns(notifications));
    }

    return response;
}

}

```

Листинг Б.10 – Файл ProfileServiceImpl.java

```

@Service
public class ProfileServiceImpl implements ProfileService {

    @Autowired
    private UserRepository userRepository;
    @Autowired
    private StreamerRepository streamerRepository;
    @Autowired
    private FollowerRepository followerRepository;

    @Override
    public ProfileResponse getProfile(Integer userId) {
        ProfileResponse profileResponse = new ProfileResponse();

        if (userId != null) {
            Optional<Users> user =
userRepository.findById(userId);
            user.ifPresent(value -> {
                profileResponse.setLogin(value.getLogin());

profileResponse.setChannelName(value.getStreamer().getChannelNam
e());

profileResponse.setIcon(value.getStreamer().getIcon());
                profileResponse.setFollowers(0);
            });
        } else {
            profileResponse.setErrorResponse(new ErrorResponse(
                Constants.BUSINESS_ERROR_CODE,
                Constants.BusinessSubCode.NOT_AUTHORIZED,
                "Необходимо авторизоваться",
                "Чтобы получить доступ к профилю необходимо
выполнить вход или зарегистрироваться"
            ));
        }
    }
}

```

```

        return profileResponse;
    }

    @Override
    public BaseResponse editProfile(Integer userId, String
login, String channelName) {
        BaseResponse response = new BaseResponse();
        Users user = new Users();
        user.setLogin(login);

        Optional<Users> userFromDB =
userRepository.findOne(Example.of(user));
        if (userFromDB.isPresent()) {
            if (userFromDB.get().getLogin().equals(login)) {
                response.setErrorResponse(new ErrorResponse(
                    Constants.BUSINESS_ERROR_CODE,
Constants.BusinessSubCode.USER_EXIST_SUB_CODE,
                    "Логин занят"
                ));
            }
            return response;
        }

        Streamer streamer = new Streamer();
        streamer.setChannelName(channelName);
        user.setStreamer(streamer);

        userFromDB = userRepository.findOne(Example.of(user));
        if (userFromDB.isPresent()) {
            if
(userFromDB.get().getStreamer().getChannelName().equals(channelN
ame)) {
                response.setErrorResponse(new ErrorResponse(
                    Constants.BUSINESS_ERROR_CODE,
Constants.BusinessSubCode.USER_EXIST_SUB_CODE,
                    "Название канала занято"
                ));
            }
            return response;
        }

        user.setUserId(userId);

        streamer.setStreamerId(userRepository.findStreamerIdByUserId(use
rId));
        streamerRepository.save(streamer);
        userRepository.save(user);

        return response;
    }

```

```

@Override
public StreamersResponse getFollowers(Integer userId) {
    Integer streamerId =
userRepository.findStreamerIdByUserId(userId);
    List<Follower> followers =
followerRepository.findFollowersByStreamerId(streamerId);

    StreamersResponse response = new StreamersResponse();
    List<StreamerResponse> streamers = new ArrayList<>();
    for (Follower follower : followers) {
        Optional<Streamer> streamer =
streamerRepository.findById(follower.getFollowerId().getFollower
Id());
        streamer.ifPresent(value -> streamers.add(
            DataModelToResponseModel.toStreamer(value)
        ));
    }

    if (streamers.isEmpty()) {
        response.setErrorResponse(new ErrorResponse(
            Constants.BUSINESS_ERROR_CODE,
            Constants.BusinessSubCode.EMPTY_LIST,
            "У Вас пока что нет подписчиков"
        ));
    } else {
        response.setItems(streamers);
    }

    return response;
}
}

```

Листинг Б.11 – Файл StreamServiceImpl.java

```

@Service
public class StreamServiceImpl implements StreamService{

    @Autowired
    private StreamRepository streamRepository;
    @Autowired
    private FollowerRepository followerRepository;
    @Autowired
    private UserRepository userRepository;
    @Autowired
    private StreamerRepository streamerRepository;
    @Autowired
    private GameRepository gameRepository;

    @Override
    public StreamResponse getStream(Integer userId, Integer
streamId) {
        StreamResponse response;
    }
}

```

```

        Optional<Stream> oStream =
streamRepository.findById(streamId);

        if (oStream.isPresent()) {
            Stream stream = oStream.get();

            Boolean isFollowing =
followerRepository.isFollowing(stream.getStreamer().getStreamerI
d(), userId) > 0;

            response = new StreamResponse(
                stream.getStreamId(),
                stream.getUrl(),
                stream.getName(),
                stream.getIcon(),
                stream.getViewers(),
                stream.getGame().getName(),
                stream.getStreamer().getChannelName(),
                stream.getStreamer().getIcon(),

followerRepository.followersCount(stream.getStreamer().getStream
erId()),

                isFollowing,
                stream.getStartTime()
            );
        } else {
            response = new StreamResponse(new ErrorResponse(
                Constants.BUSINESS_ERROR_CODE,
                Constants.BusinessSubCode.NOT_FOUND,
                "Трансляция не найдена. Возможно она уже
закончилась"
            ));
        }

        return response;
    }

    @Override
    public BaseResponse follow(Integer userId, Integer streamId,
Boolean isFollowing) {
        BaseResponse response = new BaseResponse();
        Optional<Stream> stream =
streamRepository.findById(streamId);
        if (stream.isPresent()) {
            Integer streamerId =
stream.get().getStreamer().getStreamerId();
            Integer followerId =
userRepository.findStreamerIdByUserId(userId);
            Follower follower = new Follower();
            follower.setFollowerId(new FollowerId(streamerId,
followerId));
            if (isFollowing) {
                followerRepository.save(follower);
            }
        }
    }

```

```

        } else {
            followerRepository.delete(follower);
        }
    } else {
        response.setErrorResponse(new ErrorResponse(
            Constants.BUSINESS_ERROR_CODE,
            Constants.BusinessSubCode.NOT_FOUND,
            "Трансляция не найдена. Возможно она уже
закончилась"
        ));
    }
    return response;
}

@Override
public BaseResponse addStream(Integer userId, String
streamName, String gameName) {
    Stream stream = new Stream();
    stream.setName(streamName);
    stream.setStartTime(LocalDate.now().toString());
    stream.setViewers(0);

    Optional<Streamer> streamer =
streamerRepository.findById(userRepository.findStreamerIdByUserI
d(userId));
    stream.setStreamer(streamer.get());

    Game game = gameRepository.findGameByName(gameName);
    stream.setGame(game);

    streamRepository.save(stream);
    return new BaseResponse();
}
}

```

Приложение В
ИСХОДНЫЙ КОД МОБИЛЬНОГО ПРИЛОЖЕНИЯ
(рекомендуемое)

Листинг В.1 – Файл BaseActivity.kt

```

abstract class BaseActivity<T : BaseViewModel> :
    AppCompatActivity(), BaseLifecycleOwner {

    abstract val viewModel: T

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        initObservers()
    }

    override fun onSupportNavigateUp(): Boolean {
        onBackPressed()
        return super.onSupportNavigateUp()
    }

    private fun initObservers() {
        viewModel.errorShow.observe {
            AlertDialog.getDialog(
                it.errorMsg ?:
resources.getString(R.string.common_network_error),
                when (it.action) {
                    ErrorAction.BACK -> {
                        object :
AlertDialog.MessageDialogDismissListener {
                            override fun onDismiss() {
                                onBackPressed()
                            }
                        }
                    }
                    else -> null
                }
            ).show(supportFragmentManager,
AlertDialog::class.simpleName)
        }
    }
}

```

Листинг В.2 – Файл BaseFragment.kt

```

abstract class BaseFragment<T : BaseViewModel> : Fragment(),
    BaseLifecycleOwner {

    abstract val viewModel: T

    override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
        super.onCreateView(view, savedInstanceState)
        initObservers()
    }
}

```

```

        private fun initObservers() {
            viewModel.errorShow.observe {
                AlertDialog.getDialog(
                    it.errorMsg ?:
resources.getString(R.string.common_network_error)
                ).show(childFragmentManager,
AlertDialog::class.simpleName)
            }
        }
    }
}

```

Листинг В.3 – Файл BaseLifecycleOwner.kt

```

interface BaseLifecycleOwner : LifecycleOwner {
    fun <T : Any?> Data<T>.observe(action: (T) -> Unit) {
        this.liveData.observe(this@BaseLifecycleOwner, Observer
{ action.invoke(it) })
    }

    fun <T> DisplayableData<T>.observe(action: (data: T?,
isVisible: Boolean) -> Unit) {
        this.liveData.observe(
            this@BaseLifecycleOwner,
            Observer { action.invoke(it.first, it.second) }
        )
    }

    fun <T> Action<T>.observe(action: (T) -> Unit) {
        this.observe(this@BaseLifecycleOwner, Observer {
action.invoke(it) })
    }
}

```

Листинг В.4 – Файл BaseViewModel.kt

```

open class BaseViewModel : ViewModel() {

    private val defApiErrorHandler: (error: ErrorResponseMo) ->
Unit = {
        onError(ErrorEvent(it.errorMsg))
    }

    private val defNetworkErrorHandler: () -> Unit = {
        onError(ErrorEvent(action = ErrorAction.BACK))
    }

    internal val errorShow = Action<ErrorEvent>()

    protected open fun onError(error: ErrorEvent) {
        errorShow.call(error)
    }
}

```

```

protected fun <T> Data<T>.set(data: T) {
    internalSet(data)
}

protected fun <T> DisplayableData<T>.show(data: T) {
    internalSet(data to true)
}

protected fun DisplayableData<Unit>.show() {
    internalSet(Unit to true)
}

protected fun <T> DisplayableData<T>.hide() {
    internalSet(null to false)
}

protected fun <T> Action<T>.call(data: T) {
    internalCall(data)
}

protected fun Action<Unit>.call() {
    internalCall(Unit)
}

fun <T : BaseResponseMo> launch(
    block: suspend () -> T,
    loading: (Boolean) -> Unit,
    successHandler: (T) -> Unit,
    apiErrorHandler: ((ErrorResponseMo, (ErrorResponseMo) ->
Unit) -> Unit)? = null,
    networkErrorHandler: (() -> Unit)? = null
) {
    viewModelScope.launch {
        try {
            loading.invoke(true)
            val response = block.invoke()
            response.errorResponse?.let {
                apiErrorHandler?.invoke(it,
defApiErrorHandler) ?: defApiErrorHandler.invoke(it)
            } ?: successHandler.invoke(response)
        } catch (throwable: Throwable) {
            if (throwable !is CancellationException) {
                networkErrorHandler?.invoke() ?:
defNetworkErrorHandler.invoke()
            }
        } finally {
            loading.invoke(false)
        }
    }
}
}

```

Листинг В.5 – Файл FollowingFragment.kt

```
class FollowingFragment : BaseFragment<FollowingViewModel>() {

    override val viewModel: FollowingViewModel by lazy {
        viewModel(
            FollowingViewModel(FollowingRepositoryImpl(RetrofitProvider.createApi(FollowingApi::class.java)))
        )
    }

    private var binding: FragmentFollowingBinding by
        viewBinding()

    private val streamerAdapter = StreamerAdapter()
    private val gameAdapter = FavoriteGameAdapter(
        onGameClick = { viewModel.onGameClick(it) }
    )

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentFollowingBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.rvStreamers.apply {
            adapter = streamerAdapter
            addItemDecoration(
                SpaceItemDecoration(
                    spaceHorizontal =
                        resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

        binding.rvFavoriteGames.apply {
            adapter = gameAdapter
            addItemDecoration(
                SpaceItemDecoration(
                    spaceHorizontal =
                        resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }
    }
}
```

```

        viewModel.onViewCreated()

        initObservers()
    }

    private fun initObservers() {
        viewModel.streamers.observe { data, isVisible ->
            data?.let { streamerAdapter.set(it) }
            binding.vgStreamers.visible = isVisible
        }
        viewModel.favoriteGames.observe { data, isVisible ->
            data?.let { gameAdapter.set(it) }
            binding.vgFavoriteGames.visible = isVisible
        }
        viewModel.showGameInfo.observe {
            GameInfoBottomDialog.getDialog(it).apply {
                setListener(object :
GameInfoBottomDialog.GameInfoDialogListener {
                    override fun onFollowClick(gameId: Int,
isFavoriteGame: Boolean) {
                        viewModel.onFollowClick(gameId,
isFavoriteGame)
                    }
                })
            }.show(childFragmentManager,
GameInfoBottomDialog::class.simpleName)
        }
    }
}

```

Листинг В.6 – Файл FollowingViewModel.kt

```

class FollowingViewModel(private val repository:
FollowingRepository) : BaseViewModel() {

    val streamers = DisplayableData<List<Streamer>>()
    val favoriteGames = DisplayableData<List<Game>>()
    val showGameInfo = Action<GameItem>()

    fun onViewCreated() {
        getGames()
        getStreamers()
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun getStreamers() {
        launch(
            block = { repository.getStreamers() },
            loading = { onStreamersLoading(it) },
            successHandler = { onStreamersGet(it) }
        )
    }
}

```

```

        )
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun onStreamersLoading(isLoading: Boolean) {
        if (isLoading) streamers.show(
            listOf(
                StreamerLoader,
                StreamerLoader,
                StreamerLoader,
                StreamerLoader
            )
        )
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun onStreamersGet(streamers: StreamersResponseMo) {
        this.streamers.show(streamers.items.toStreamers())
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun getGames() {
        launch(
            block = { repository.getFavoriteGames() },
            loading = { onFavoriteGamesLoading(it) },
            successHandler = { onFavoriteGamesGet(it) }
        )
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun onFavoriteGamesLoading(isLoading: Boolean) {
        if (isLoading) favoriteGames.show(listOf(GameLoader,
        GameLoader, GameLoader))
    }

    @VisibleForTesting(otherwise = PRIVATE)
    fun onFavoriteGamesGet(games: GamesResponseMo) {
        favoriteGames.show(games.items.toGames())
    }

    fun onGameClick(game: GameItem) {
        showGameInfo.call(game)
    }

    fun onFollowClick(gameId: Int, isFavoriteGame: Boolean) {
    }

}

```

Листинг В.7 – Файл HomeFragment.kt

```

class HomeFragment : BaseFragment<HomeViewModel>() {

    override val viewModel: HomeViewModel by lazy {

viewModel(HomeViewModel(HomeRepositoryImpl(RetrofitProvider.crea
teApi(HomeApi::class.java))))
    }

    private var binding: FragmentHomeBinding by viewBinding()

    private val streamerAdapter = StreamerAdapter()
    private val streamAdapter = StreamAdapter {
viewModel.onStreamClick(it) }

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentHomeBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.rvStreams.apply {
            adapter = streamAdapter
            addItemDecoration(
                SpaceItemDecoration(
                    spaceHorizontal =
resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

        binding.rvStreamersOnline.apply {
            adapter = streamerAdapter
            addItemDecoration(
                SpaceItemDecoration(
                    spaceHorizontal =
resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

        viewModel.onViewCreated()

        initObservers()
    }
}

```

```

    }

    private fun initObservers() {
        viewModel.streams.observe { data, isVisible ->
            data?.let { streamAdapter.set(it) }
            binding.vgStreams.visible = isVisible
        }
        viewModel.streamers.observe { data, isVisible ->
            data?.let { streamerAdapter.set(it) }
            binding.vgStreamers.visible = isVisible
        }
        viewModel.goToStream.observe {

startActivity(WatchStreamActivity.getIntent(requireContext(),
it.streamId))
        }
    }
}

```

Листинг В.8 – Файл HomeViewModel.kt

```

class HomeViewModel(private val repository: HomeRepository) :
BaseViewModel() {

    val streams = DisplayableData<List<Stream>>()
    val streamers = DisplayableData<List<Streamer>>()
    val goToStream = Action<Stream>()

    fun onCreateView() {
        getStreams()
        getStreamers()
    }

    @VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
    fun getStreams() {
        launch(
            block = { repository.getStreams() },
            loading = { },
            successHandler = { onStreamsGet(it) }
        )
    }

    @VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
    fun onStreamsGet(streams: StreamsResponseMo) {
        this.streams.show(streams.items.toStreams())
    }

    @VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
    fun getStreamers() {
        launch(
            block = { repository.getStreamers() },
            loading = { onStreamersLoading(it) },

```

```

        successHandler = { onStreamersGet(it) }
    )
}

@VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
fun onStreamersLoading(isLoading: Boolean) {
    if (isLoading) streamers.show(listOf(StreamerLoader,
StreamerLoader, StreamerLoader))
}

@VisibleForTesting(otherwise = VisibleForTesting.PRIVATE)
fun onStreamersGet(streamers: StreamersResponseMo) {
    this.streamers.show(streamers.items.toStreamers())
}

fun onStreamClick(game: Stream) {
    goToStream.call(game)
}
}

```

Листинг В.9 – Файл WatchStreamActivity.kt

```

class WatchStreamActivity : BaseActivity<WatchStreamViewModel>()
{
    companion object {
        private const val KEY_STREAM_ID = "KEY_STREAM_ID"
        fun getIntent(context: Context, streamId: Int) =
            Intent(context,
WatchStreamActivity::class.java).apply {
                putExtra(KEY_STREAM_ID, streamId)
            }
    }

    override val viewModel: WatchStreamViewModel by lazy {
        viewModel(
WatchStreamViewModel(WatchStreamRepositoryImpl(RetrofitProvider.
createApi(StreamApi::class.java)))
        )
    }

    private lateinit var binding: ActivityWatchStreamBinding

    private val messageAdapter = MessageAdapter()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding =
ActivityWatchStreamBinding.inflate(layoutInflater)
        setContentView(binding.root)
    }
}

```

```

        binding.rvMessage.apply {
            adapter = messageAdapter
            addItemDecoration(
               SpaceItemDecoration(
                    spaceVertical =
resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

viewModel.onGetStream(intent.getIntExtra(KEY_STREAM_ID,
-1))

        initObservers()
        initListeners()
    }

    private fun initObservers() {
        viewModel.stream.observe {
            binding.svPlayer.player = player.apply {
                setMediaItem(MediaItem.fromUri(it.url))
                prepare()
                play()
            }
            binding.tvName.text = it.name
            it.channelIcon?.let { url ->
                binding.ivChannelIcon.loadImage(url,
transformMode = TransformMode.CIRCLE_CROP)
            } ?:
binding.ivChannelIcon.setImageDrawable(AppCompatResources.getDrawa
ble(this, R.drawable.ic_user_default))
            binding.tvViewers.text = "${it.viewers} зрителей"
            binding.tvTime.text = it.startTime
            binding.tvChannelName.text = it.channelName
            binding.tvFollowersCount.text = getString(
                R.string.profile_x_followers,

NumberFormat.getNumberInstance(Locale.FRANCE).format(it.follower
sCount)
            )
        }
        viewModel.followState.observe {
            binding.btnFollow.text = if (it) "Отписаться" else
"Подписаться"
        }
    }

    private fun initListeners() {
        binding.btnFollow.setOnClickListener {
            viewModel.onFollowClick()
        }
    }
}

```

```
}
```

Листинг В.10 – Файл WatchStreamViewModel.kt

```
class WatchStreamViewModel(private val repository:
WatchStreamRepository): BaseViewModel() {

    private var streamId: Int = -1
    private var isFollowing = false

    val stream = Data<Stream>()
    val followState = Data<Boolean>()

    fun onGetStream(streamId: Int) {
        this.streamId = streamId
        launch(
            block = { repository.getStream(streamId) },
            loading = { },
            successHandler = { onStreamGet(it) }
        )
    }

    private fun onStreamGet(stream: StreamResponse) {
        this.stream.set(stream.toStream())
        isFollowing = stream.isFollowing
        followState.set(stream.isFollowing)
    }

    fun onFollowClick() {
        follow()
    }

    private fun follow() {
        launch(
            block = { repository.follow(streamId, !isFollowing)
},
            loading = { },
            successHandler = { onFollow() }
        )
    }

    private fun onFollow() {
        isFollowing = !isFollowing
        followState.set(isFollowing)
    }

}
```

Листинг В.11 – Файл CompleteStreamActivity.kt

```
class CompleteStreamActivity :
BaseActivity<CompleteStreamViewModel>() {
```

```

        companion object {
            private const val KEY_STREAM_ID = "KEY_STREAM_ID"
            fun getIntent(context: Context, streamId: Int) =
                Intent(context,
CompleteStreamActivity::class.java).apply {
                    putExtra(KEY_STREAM_ID, streamId)
                }
        }

        override val viewModel: CompleteStreamViewModel by lazy {
            viewModel(
CompleteStreamViewModel(CompleteStreamRepositoryImpl(RetrofitPro
vider.createApi(StreamApi::class.java)))
            )
        }

        private lateinit var binding: ActivityCompleteStreamBinding

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            binding =
ActivityCompleteStreamBinding.inflate(layoutInflater)
            setContentView(binding.root)

viewModel.onViewCreated(intent.getIntExtra(KEY_STREAM_ID, -1))

            initObservers()
            initListeners()
        }

        private fun initObservers() {
            viewModel.streamInfo.observe {
                binding.tvViewers.text = it.viewers.toString()
                binding.tvTime.text = it.time
                binding.tvGame.text = it.gameName
                binding.ivGameIcon.loadImage(it.gameIcon,
transformMode = TransformMode.CIRCLE_CROP)
            }
        }

        private fun initListeners() {
            binding.ivClose.setOnClickListener {
                viewModel.onCloseClick()
            }
        }
    }
}

```

Листинг В.12 – Файл CompleteStreamViewModel.kt

```
class CompleteStreamViewModel(private val repository:
CompleteStreamRepository): BaseViewModel() {

    val streamInfo = Data<StreamInfo>()
    val goToHome = Action<Unit>()

    fun onViewCreated(streamId: Int) {
        launch(
            block = { repository.getStreamInfo(streamId) },
            loading = { },
            successHandler = { onInfoGet(it) },
        )
    }

    private fun onInfoGet(streamInfoResponse:
StreamInfoResponse) {
        streamInfo.set(streamInfoResponse.toStreamInfo())
    }

    fun onCloseClick() {
        goToHome.call()
    }

}
```

Листинг В.13 – Файл NotificationFragment.kt

```
class NotificationFragment :
BaseFragment<NotificationViewModel>() {

    override val viewModel: NotificationViewModel by lazy {
        viewModel(
            NotificationViewModel(
                NotificationRepositoryImpl(RetrofitProvider.createApi(Notificati
onApi::class.java))
            )
        )
    }

    private var binding: FragmentNotificationBinding by
viewBinding()

    private val notificationAdapter = NotificationAdapter()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
```

```

        binding = FragmentNotificationBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.rvNotifications.apply {
            adapter = notificationAdapter
            addItemDecoration(
                SpaceItemDecoration(
                    spaceVertical =
resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

        viewModel.onViewCreated()

        initObservers()
    }

    private fun initObservers() {
        viewModel.loader.observe { _, isVisible ->
            binding.sfLoader.visible = isVisible
        }
        viewModel.notifications.observe { data, isVisible ->
            data?.let { notificationAdapter.set(it) }
            binding.rvNotifications.visible = isVisible
        }
        viewModel.error.observe { data, isVisible ->
            data?.let {
                binding.evError.setMessage(it.errorMsg ?:
getString(R.string.common_network_error))
            }
            binding.evError.visible = isVisible
        }
    }
}

```

Листинг В.14 – Файл NotificationViewModel.kt

```

class NotificationViewModel(private val repository:
NotificationRepository) : BaseViewModel() {

    val loader = DisplayableData<Unit>()
    val notifications = DisplayableData<List<Notification>>()
    val error = DisplayableData<ErrorEvent>()

    fun onViewCreated() {
        launch(

```

```

        block = { repository.getNotifications() },
        loading = { onNotificationsLoading(it) },
        successHandler = { onNotificationsGet(it) }
    )
}

private fun onNotificationsLoading(isLoading: Boolean) {
    if (isLoading) loader.show()
    else loader.hide()
}

private fun onNotificationsGet(notifications:
NotificationsResponseMo) {
    this.notifications.show(notifications.toNotifications())
}

override fun onError(error: ErrorEvent) {
    this.error.show(error)
}
}

```

Листинг В.15 – Файл ProfileFragment.kt

```

class ProfileFragment : BaseFragment<ProfileViewModel>(),
    ProfileAlertDialog.ProfileActionListener {

    override val viewModel: ProfileViewModel by lazy {
        viewModel(
            ProfileViewModel(
                ProfileRepositoryImpl(
                    RetrofitProvider.createApi(ProfileApi::class.java),
                    RetrofitProvider.createApi(AuthorizationApi::class.java)
                )
            )
        )
    }

    private var binding: FragmentProfileBinding by viewBinding()

    private lateinit var childFM: FragmentManager

    private var dialog: ProfileAlertDialog? = null

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        binding = FragmentProfileBinding.inflate(inflater)
        return binding.root
    }
}

```

```

    }

    override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
        super.onCreate(savedInstanceState)

        childFM = childFragmentManager

        observeData()
        observeActions()
        initListeners()
    }

    private fun observeData() {
        viewModel.loader.observe { _, isVisible ->
            binding.sfLoaderInfo.visible = isVisible
            binding.sfLoaderActions.visible = isVisible
        }
        viewModel.profile.observe { data, isVisible ->
            data?.let {
                binding.tvChannelName.text = it.channelName
                it.icon?.let { icon ->
                    binding.ivChannelIcon.loadImage(icon,
transformMode = TransformMode.CIRCLE_CROP)
                } ?: binding.ivChannelIcon.setImageDrawable(
AppCompatResources.getDrawable(requireContext(),
R.drawable.ic_user_default)
                )
                binding.tvChannelFollowers.text = getString(
                    R.string.profile_x_followers,

NumberFormat.getNumberInstance(Locale.FRANCE).format(it.follower
s)
                )
            }
            binding.clInfo.visible = isVisible
            binding.llActions.visible = isVisible
        }
        viewModel.error.observe { data, isVisible ->
            data?.let {
                val (error, isButtonVisible) = it
                binding.evError.setTitle(error.errorTitle)
                binding.evError.setMessage(error.errorMsg)
            }
        }
        binding.evError.setButtonsVisible(isButtonVisible)
    }
    binding.gProfile.visible = !isVisible
    binding.evError.visible = isVisible
}

    private fun observeActions() {

```

```

        viewModel.showLoginDialog.observe {
            dialog = ProfileAlertDialog.getLoginDialog()
            dialog?.setListener(this@ProfileFragment)
            dialog?.show(childFM,
ProfileAlertDialog::class.simpleName)
        }
        viewModel.showSignUpDialog.observe {
            dialog = ProfileAlertDialog.getSignUpDialog()
            dialog?.setListener(this@ProfileFragment)
            dialog?.show(childFM,
ProfileAlertDialog::class.simpleName)
        }
        viewModel.showEditDialog.observe {
            dialog = ProfileAlertDialog.getEditDialog(it.icon,
it.login, it.channelName)
            dialog?.setListener(this@ProfileFragment)
            dialog?.show(childFM,
ProfileAlertDialog::class.simpleName)
        }
        viewModel.showErrorMsgInDialog.observe {
            dialog?.showErrorMsg(it)
        }
        viewModel.showValidationError.observe {
            dialog?.showValidationError(it)
        }
        viewModel.hideDialog.observe {
            dialog?.dismiss()
            dialog = null
        }
        viewModel.goToFollowers.observe {

startActivity(FollowersActivity.getIntent(requireContext(),
"Подписчики"))
        }
    }

    private fun initListeners() {
        binding.evError.btnFirst.setOnClickListener {
            viewModel.onErrorLoginClick()
        }
        binding.evError.btnSecond.setOnClickListener {
            viewModel.onErrorSignUpClick()
        }
        binding.abvFollowers.setOnClickListener {
            viewModel.onFollowersClick()
        }
        binding.ivEdit.setOnClickListener {
            viewModel.onEditClick()
        }
    }

    override fun onSaveClick(login: String, channelName: String)
{

```

```

        viewModel.onSaveClick(login, channelName)
    }

    override fun onLoginClick(login: String, password: String) {
        viewModel.onLoginClick(login, password)
    }

    override fun onSignUpClick(login: String, channelName:
String, password: String) {
        viewModel.onSignUpClick(login, channelName, password)
    }
}

```

Листинг В.16 – Файл ProfileViewModel.kt

```

class ProfileViewModel(private val repository:
ProfileRepository) : BaseViewModel() {

    private lateinit var profileInfo: ProfileResponseMo

    val loader = DisplayableData<Unit>()
    val profile = DisplayableData<ProfileResponseMo>()

    /* first - ошибка, second - флаг для кнопок */
    val error = DisplayableData<Pair<Error, Boolean>>()

    val showErrorMsgInDialog = Action<String>()
    val showValidationError = Action<List<Pair<String,
String>>>()
    val showLoginDialog = Action<Unit>()
    val showSignUpDialog = Action<Unit>()
    val showEditDialog = Action<ProfileResponseMo>()
    val hideDialog = Action<Unit>()
    val goToFollowers = Action<Unit>()

    init {
        getProfile()
    }

    private fun getProfile() {
        error.hide()
        profile.hide()
        launch {
            block = { repository.getProfile() },
            loading = { onProfileLoading(it) },
            successHandler = { onProfileGet(it) },
            apiErrorHandler = { error, _ ->
                if (error.errorSubCode ==
BusinessErrorSubCode.NOT_AUTHORIZED) onErrorWithButtons(
                    error
                )
                else onErrorWithoutButtons(error)
            }
        }
    }
}

```

```

        }
    )
}

private fun onProfileLoading(isLoading: Boolean) {
    if (isLoading) loader.show()
    else loader.hide()
}

private fun onProfileGet(profileResponseMo:
ProfileResponseMo) {
    profileInfo = profileResponseMo
    profile.show(profileInfo)
}

private fun onLogin() {
    hideDialog.call()
    getProfile()
}

private fun onLoginError(error: ErrorResponseMo) {
    showErrorMsgInDialog.call(error.message)
}

private fun onSignUp() {
    hideDialog.call()
    getProfile()
}

private fun onSignUpValidationError(error: ErrorResponseMo)
{
    showValidationError.call(error.errorFields.orEmpty())
}

private fun onSave() {
    hideDialog.call()
    getProfile()
}

private fun onSaveValidationError(error: ErrorResponseMo) {
    showValidationError.call(error.errorFields.orEmpty())
}

private fun onErrorWithButtons(error: ErrorResponseMo) {
    this.error.show(error.toError() to true)
}

private fun onErrorWithoutButtons(error: ErrorResponseMo) {
    this.error.show(error.toError() to false)
}

fun onErrorLoginClick() {
    showLoginDialog.call()
}

```

```

    }

    fun onErrorSignUpClick() {
        showSignUpDialog.call()
    }

    fun onLoginClick(login: String, password: String) {
        launch(
            block = { repository.login(login, password) },
            loading = {},
            successHandler = { onLogin() },
            apiErrorHandler = { error, defApiErrorHandler ->
                if (error.errorSubCode ==
BusinessErrorSubCode.USER_NOT_EXIST_SUB_CODE) onLoginError(
                    error
                )
                else defApiErrorHandler.invoke(error)
            }
        )
    }

    fun onSignUpClick(login: String, channelName: String,
password: String) {
        launch(
            block = { repository.signUp(login, channelName,
password) },
            loading = {},
            successHandler = { onSignUp() },
            apiErrorHandler = { error, defApiErrorHandler ->
                if (error.errorCode ==
ErrorCode.VALIDATION_ERROR) onSignUpValidationError(error)
                else defApiErrorHandler.invoke(error)
            }
        )
    }

    fun onEditClick() {
        showEditDialog.call(profileInfo)
    }

    fun onFollowersClick() {
        goToFollowers.call()
    }

    fun onSaveClick(login: String, channelName: String) {
        launch(
            block = { repository.edit(login, channelName) },
            loading = {},
            successHandler = { onSave() },
            apiErrorHandler = { error, defApiErrorHandler ->
                if (error.errorCode ==
ErrorCode.VALIDATION_ERROR) onSaveValidationError(error)
                else defApiErrorHandler.invoke(error)
            }
        )
    }

```

```

        }
    )
}

}

```

Листинг В.17 – Файл FollowersActivity.kt

```

class FollowersActivity : BaseActivity<FollowersViewModel>() {

    companion object {
        private const val KEY_TITLE = "KEY_TITLE"
        fun getIntent(context: Context, title: String) =
            Intent(context, FollowersActivity::class.java).apply {
                putExtra(KEY_TITLE, title)
            }
    }

    override val viewModel: FollowersViewModel by lazy {
        viewModel(
            FollowersViewModel(FollowersRepositoryImpl(RetrofitProvider.createApi(FollowersApi::class.java)))
        )
    }

    private lateinit var binding: ActivityFollowersBinding

    private val followersAdapter = FollowersAdapter()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding =
            ActivityFollowersBinding.inflate(layoutInflater)
        setContentView(binding.root)
        setSupportActionBar(binding.toolbar)
        supportActionBar?.setDisplayHomeAsUpEnabled(true)

        binding.rvFollowers.apply {
            adapter = followersAdapter
            addItemDecoration(
               SpaceItemDecoration(
                    spaceVertical =
                        resources.getDimensionPixelOffset(R.dimen.margin_1)
                )
            )
        }

        initObservers()
        initListeners()
    }
}

```

```

private fun initObservers() {
    viewModel.loader.observe { _, isVisible ->
        binding.sfLoader.visible = isVisible
    }
    viewModel.followers.observe { data, isVisible ->
        data?.let { followersAdapter.set(it) }
        binding.rvFollowers.visible = isVisible
    }
}

private fun initListeners() {
    followersAdapter.setListener(object :
FollowersAdapter.FollowersListener {
        override fun onFollowClick(channelName: String) {
            viewModel.onFollowClick(channelName)
        }
    })
}
}

```

Листинг В.18 – Файл FollowersViewModel.kt

```

class FollowersViewModel(private val repository:
FollowersRepository) : BaseViewModel() {

    private lateinit var followersData: List<Streamer>

    val loader = DisplayableData<Unit>()
    val followers = DisplayableData<List<Streamer>>()

    init {
        launch(
            block = { repository.getFollowers() },
            loading = { onFollowersLoading(it) },
            successHandler = { onFollowersGet(it) }
        )
    }

    private fun onFollowersLoading(isLoading: Boolean) {
        if (isLoading) loader.show()
        else loader.hide()
    }

    private fun onFollowersGet(followers: StreamersResponseMo) {
        followersData = followers.items.toStreamer()
        this.followers.show(followersData)
    }

    fun onFollowClick(channelName: String) {
        followersData = followersData.map {
            if (it.channelName == channelName) {
                Streamer(

```

```

        channelName = it.channelName,
        channelIcon = it.channelIcon,
        followers = it.followers,
        isFollowing = !it.isFollowing
    )
    } else {
        it
    }
}
this.followers.show(followersData)
}
}

```

Листинг В.19 – Файл AddStreamFragment.kt

```

class AddStreamFragment : BaseFragment<AddStreamViewModel>() {

    override val viewModel: AddStreamViewModel by lazy {
        viewModel(
            AddStreamViewModel(AddStreamRepositoryImpl(RetrofitProvider.createApi(StreamApi::class.java)))
        )
    }

    private var binding: FragmentAddStreamBinding by
        viewBinding()

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentAddStreamBinding.inflate(inflater)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        binding.ivUnderlords.loadImage("https://yt3.ggpht.com/a/AGF-1797TmTKfq16nfs09c8XXO6mrGc-svnYIhXPWg=s900-c-k-c0xffffffff-no-rj-mo")

        binding.ivLineage.loadImage("https://yt3.ggpht.com/a/AATXAJyqb49Vu5kQ0R-PsiMXbx16VkVfhTi3cSDVcQV9=s900-c-k-c0x00ffffff-no-rj")

        binding.ivCallOfDuty.loadImage("https://androidtrunk.com/wp-content/uploads/2019/01/48344985_275005499869442_4889003595644534784_n-1.jpg")
    }
}

```

```

    }

}

```

Листинг В.20 – Файл AddStreamViewModel.kt

```

class AddStreamViewModel(private val repository:
AddStreamRepository): BaseViewModel() {

    val openGame = Action<Unit>()

    fun onGameClick(streamName: String, gameName: String) {
        launch(
            block = { repository.addStream(streamName, gameName)
},
            loading = { },
            successHandler = { onStreamAdd() }
        )
    }

    private fun onStreamAdd() {
        openGame.call()
    }

}

```

Листинг В.21 – Файл MainActivity.kt

```

class MainActivity : BaseActivity<MainViewModel>() {

    override val viewModel: MainViewModel by lazy {
viewModel(MainViewModel()) }

    private lateinit var binding: ActivityMainBinding

    companion object {
        fun getIntent(context: Context) = Intent(context,
MainActivity::class.java)
            .addFlags(Intent.FLAG_ACTIVITY_CLEAR_TASK)
            .addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        supportFragmentManager.beginTransaction().apply {
            replace(binding.fragmentWrapper.id, HomeFragment())
            commit()
        }

        initObservers()
    }
}

```

```

        initNavigation()
    }

    private fun initObservers() {
        viewModel.navigationItemSelect.observe {
            val (nextItemType, previousItemType) = it
            val fragment = when (nextItemType) {
                NavigationItemType.HOME -> HomeFragment()
                NavigationItemType.FOLLOWING ->
FollowingFragment()
                NavigationItemType.ADD_VIDEO ->
AddStreamFragment()
                NavigationItemType.NOTIFICATION ->
NotificationFragment()
                NavigationItemType.PROFILE -> ProfileFragment()
            }
            if (nextItemType.ordinal > previousItemType.ordinal)

supportFragmentManager.replaceWithRightAnimation(
                    binding.fragmentWrapper.id,
                    fragment
                )
            else
                supportFragmentManager.replaceWithLeftAnimation(
                    binding.fragmentWrapper.id,
                    fragment
                )
        }
    }

    private fun initNavigation() {

binding.bnvNavigationMenu.setOnNavigationItemSelectedListener {
            val previousType = when
(binding.bnvNavigationMenu.selectedItemId) {
                R.id.home -> NavigationItemType.HOME
                R.id.following -> NavigationItemType.FOLLOWING
                R.id.addVideo -> NavigationItemType.ADD_VIDEO
                R.id.notification ->
NavigationItemType.NOTIFICATION
                else -> NavigationItemType.PROFILE
            }
            when (it.itemId) {
                R.id.home ->

viewModel.onNavigationItemSelectedListener(previousType,
NavigationItemType.HOME)
                R.id.following ->

viewModel.onNavigationItemSelectedListener(previousType,
NavigationItemType.FOLLOWING)
                R.id.addVideo ->
viewModel.onNavigationItemSelectedListener(

```

```

        previousType,
        NavigationItemType.ADD_VIDEO
    )
    R.id.notification ->
        viewModel.onNavigationItemSelected(
            previousType,
            NavigationItemType.NOTIFICATION
        )
    R.id.profile ->
        viewModel.onNavigationItemSelected(previousType,
        NavigationItemType.PROFILE)
        else -> false
    }
}
}
}

```

Приложение Г
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

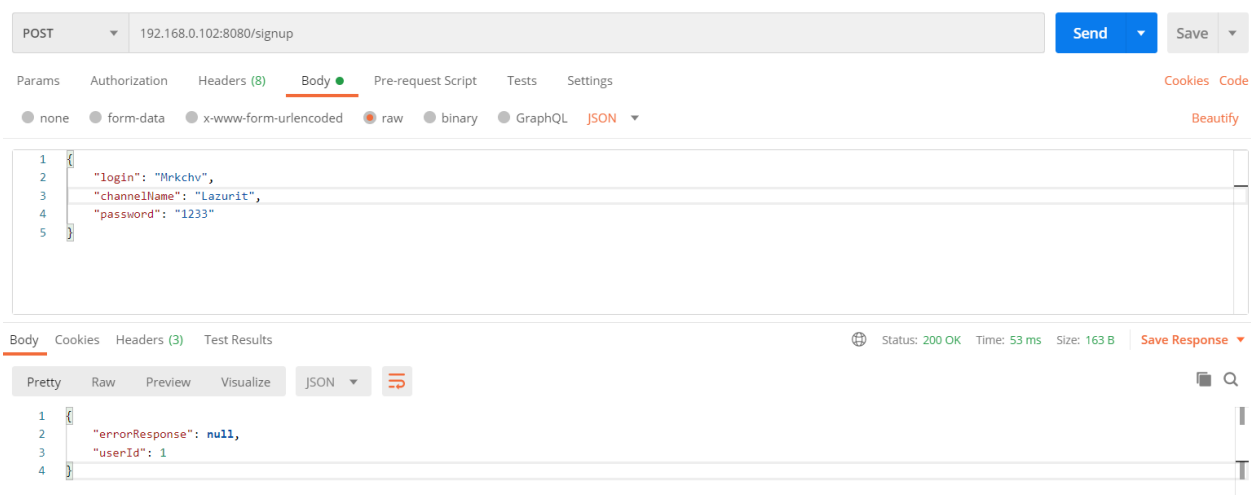


Рисунок Г.1 – Регистрация

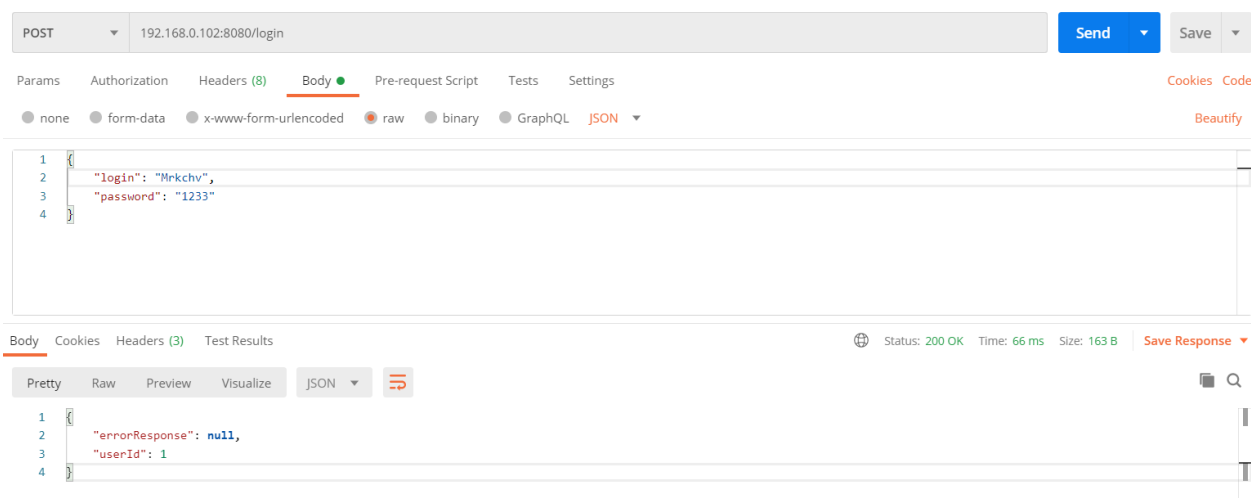


Рисунок Г.2 – Вход

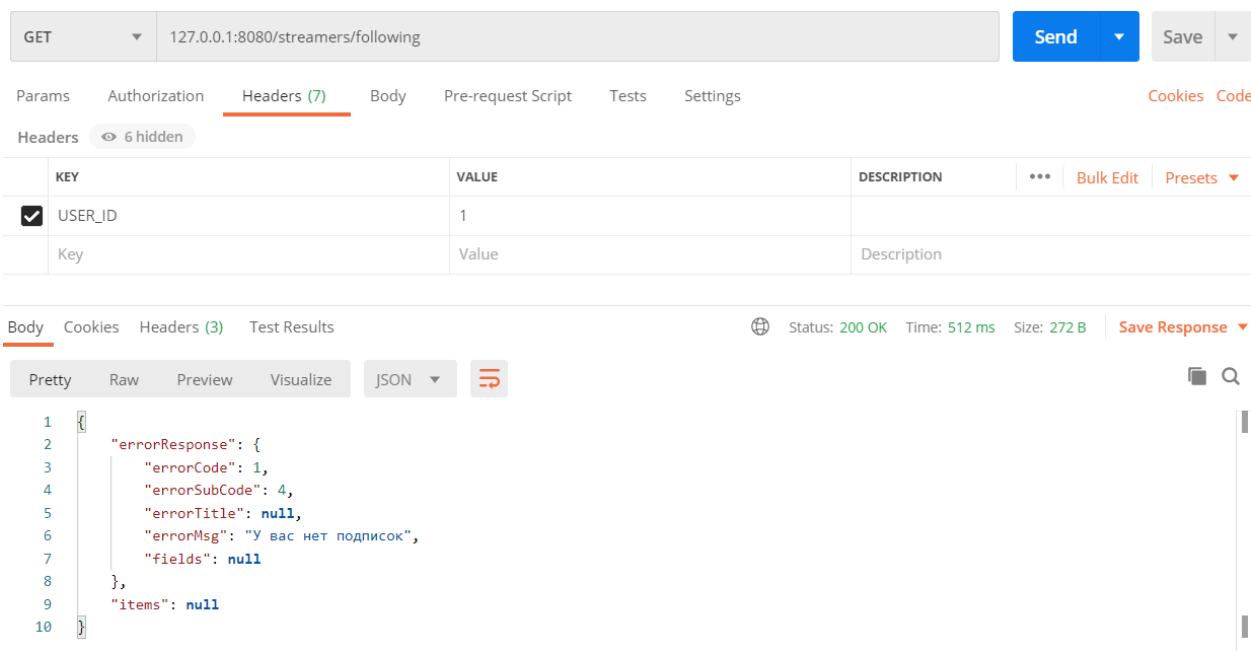


Рисунок Г.3 – Получение списка подписок

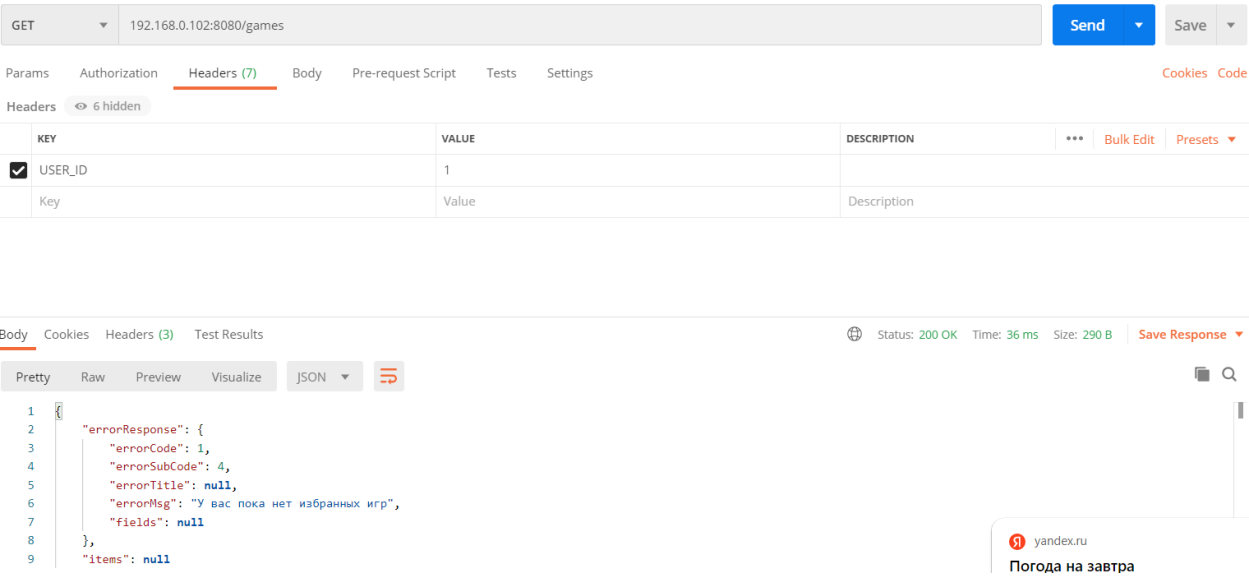


Рисунок Г.4 – Получение списка избранных игр

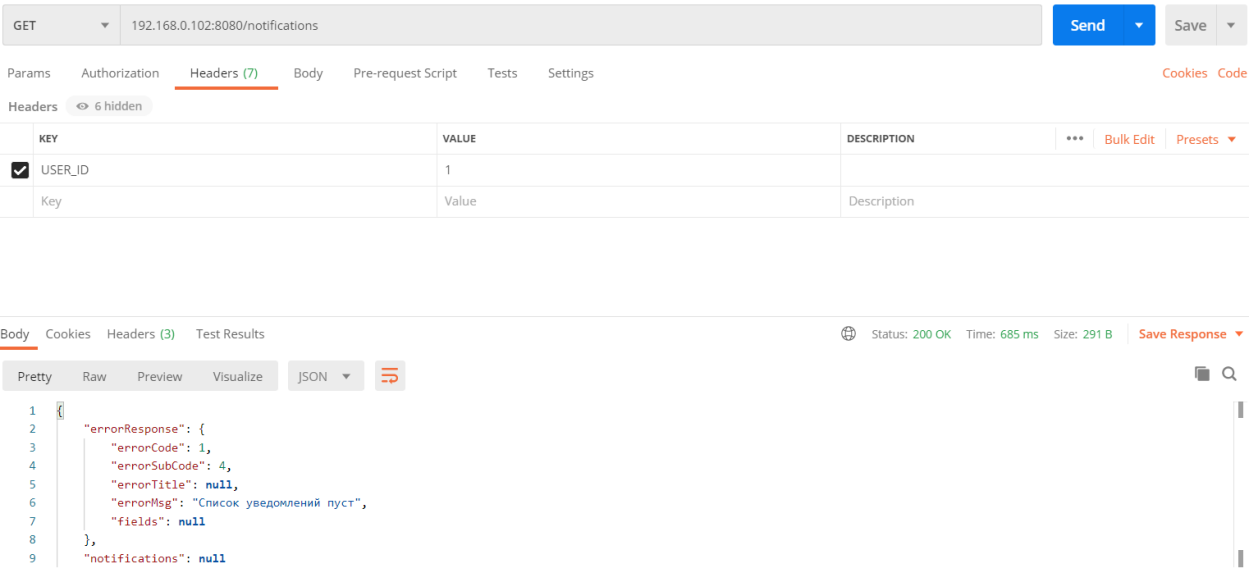


Рисунок Г.5 – Получение списка уведомлений

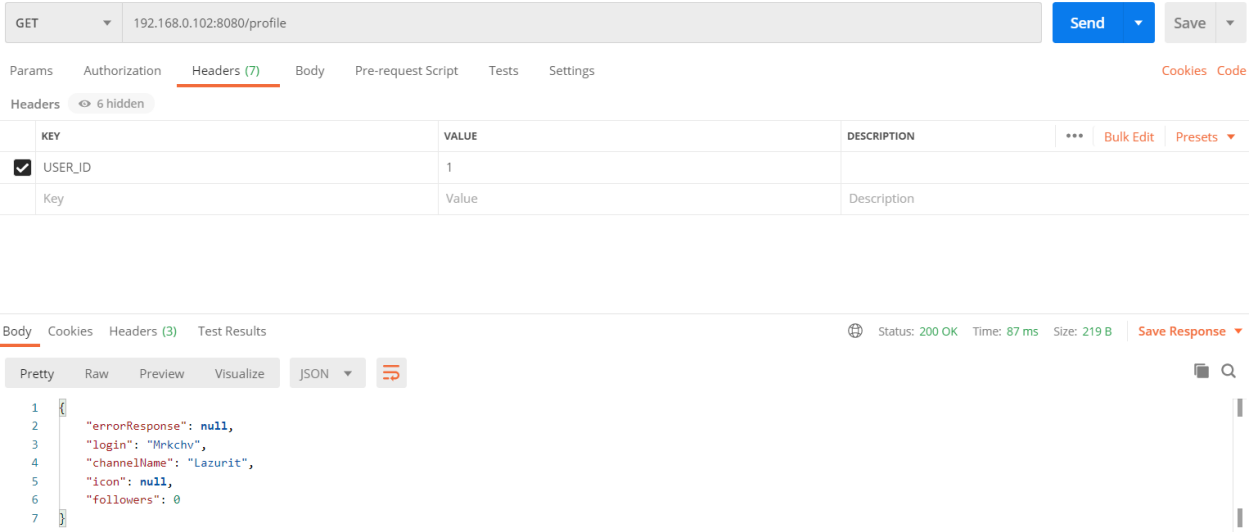


Рисунок Г.6 – Получение данных профиля

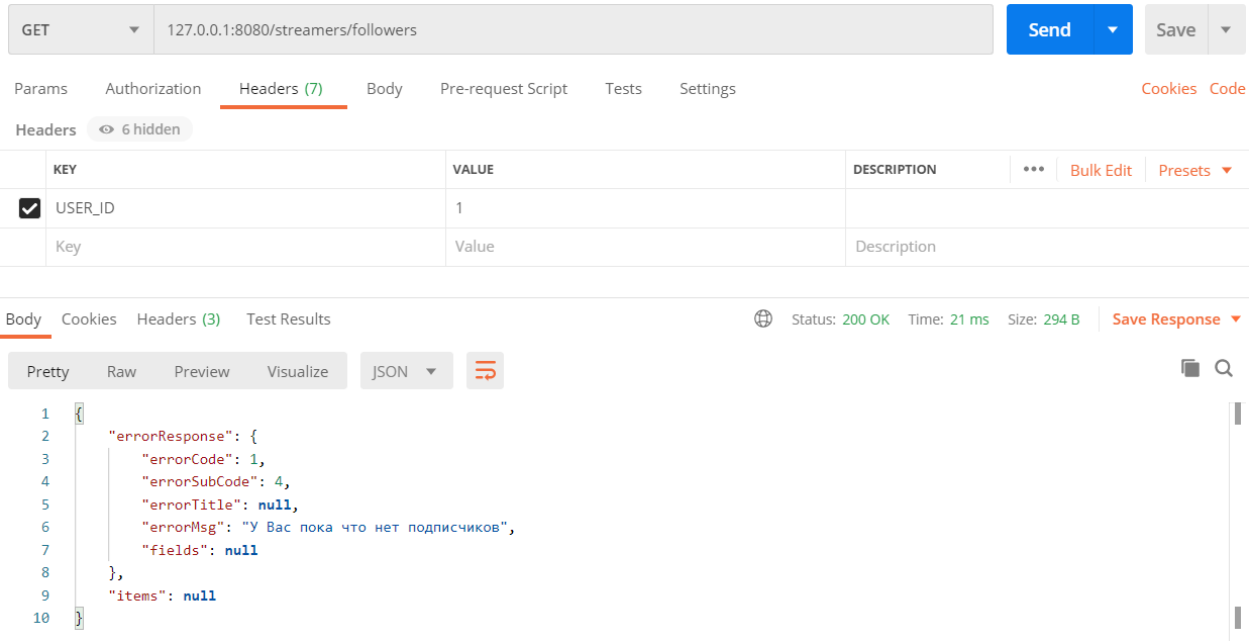


Рисунок Г.7 – Получение списка подписчиков

GET127.0.0.1:8080/stream?streamId=1

SendSave

ParamsAuthorizationHeaders (7)BodyPre-request ScriptTestsSettings

CookiesCode

Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
☒	streamId	1			
	Key	Value	Description		

BodyCookiesHeaders (3)Test Results

Status: 200 OKTime: 454 msSize: 500 BSave Response

PrettyRawPreviewVisualizeJSON

```
1 {
2   "errorResponse": {
3     "errorCode": 1,
4     "errorSubCode": 5,
5     "errorTitle": null,
6     "errorMsg": "Трансляция не найдена. Возможно она уже закончилась",
7     "fields": null
8   },
9 }
```

Рисунок Г.8 – Получение данных трансляции

POST127.0.0.1:8080/stream/add

SendSave

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

CookiesCode

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

BodyCookiesHeaders (3)Test Results

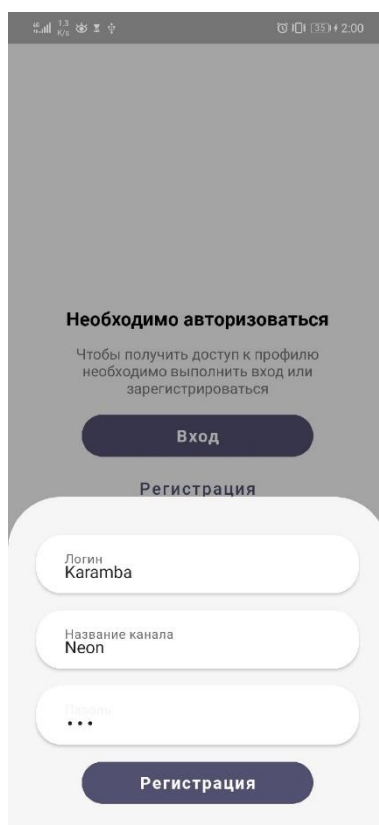
Status: 200 OKTime: 107 msSize: 152 BSave Response

PrettyRawPreviewVisualizeJSON

```
1 {
2   "streamName": "DAAAAANGE",
3   "gameName": "Lineage II Revolution"
4 }
```

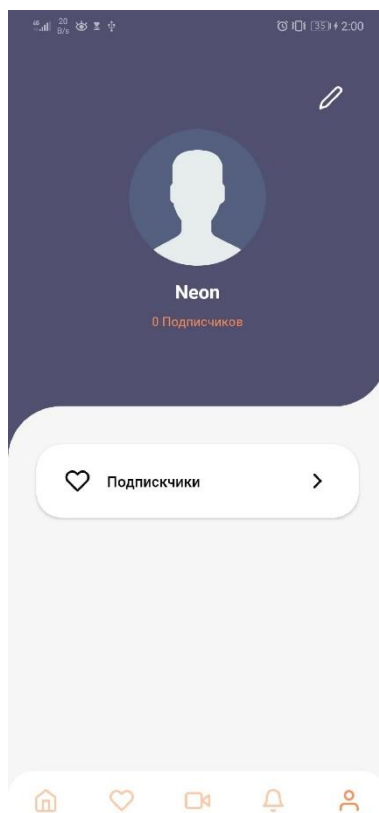
Рисунок Г.9 – Добавление трансляции

Приложение Д
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ МОБИЛЬНОГО ПРИЛОЖЕНИЯ
(рекомендуемое)



Скриншот экрана диалога регистрации. Вверху статус-бар с временем 2:00. Основной текст: «Необходимо авторизоваться. Чтобы получить доступ к профилю необходимо выполнить вход или зарегистрироваться». Кнопка «Вход» выделена. Ниже заголовок «Регистрация» и три поля ввода: «Логин» (Karamba), «Название канала» (Neon) и «Пароль» (маска * * *). Внизу кнопка «Регистрация».

Рисунок Д.1 – Диалог регистрации



Скриншот экрана профиля пользователя после регистрации. Фон темный. В центре аватарка и имя «Neon». Под именем текст «0 Подписчиков». Внизу белая кнопка «Подписчики» с сердечком и стрелкой. В самом низу — панель навигации с иконками: дом, сердце, видео, уведомление, профиль.

Рисунок Д.2 – Форма «Профиль» после регистрации

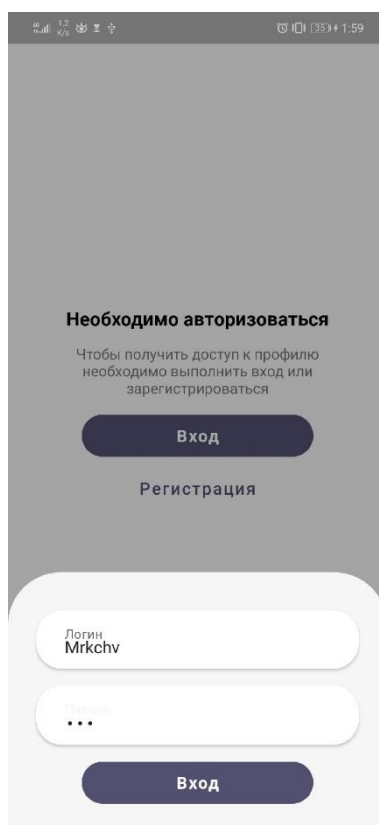


Рисунок Д.3 – Диалог входа

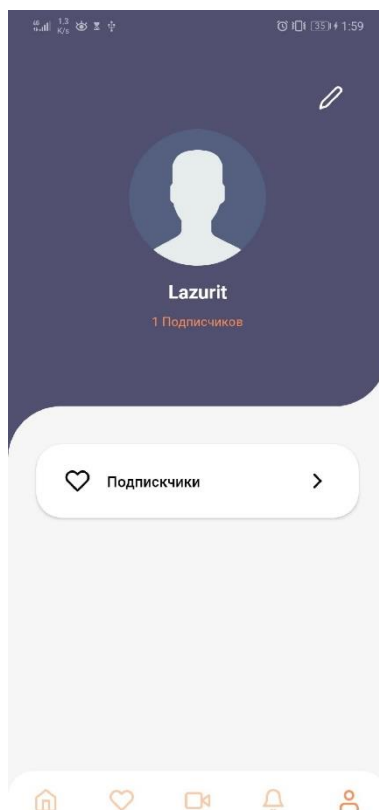


Рисунок Д.4 – Форма «Профиль» после входа

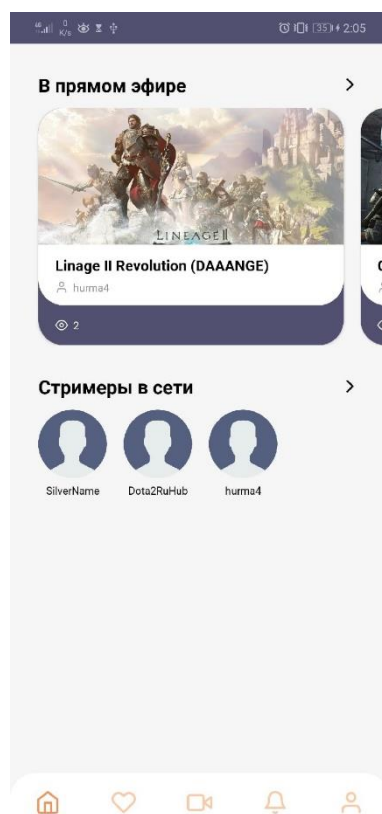


Рисунок Д.5 – Главная форма со списком трансляций

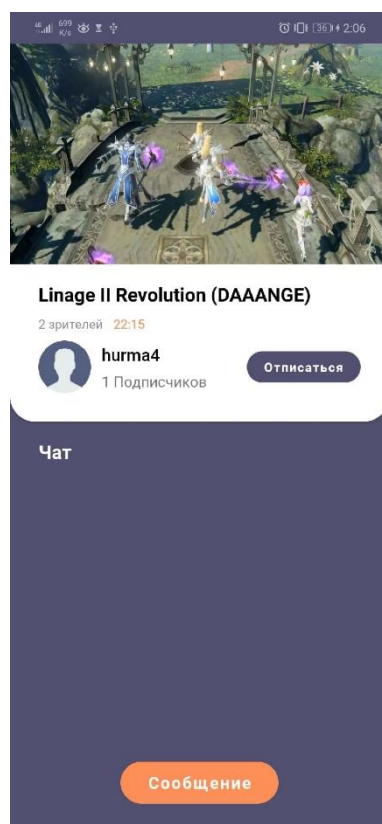


Рисунок Д.6 – Форма «Трансляция»

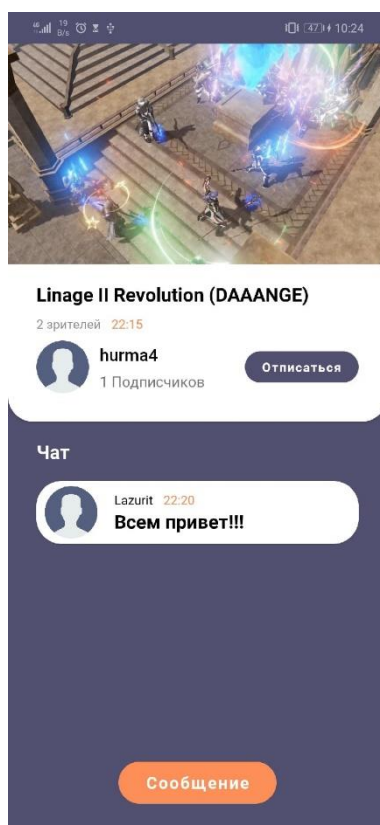


Рисунок Д.7 – Форма «Трансляция» с добавленным сообщением

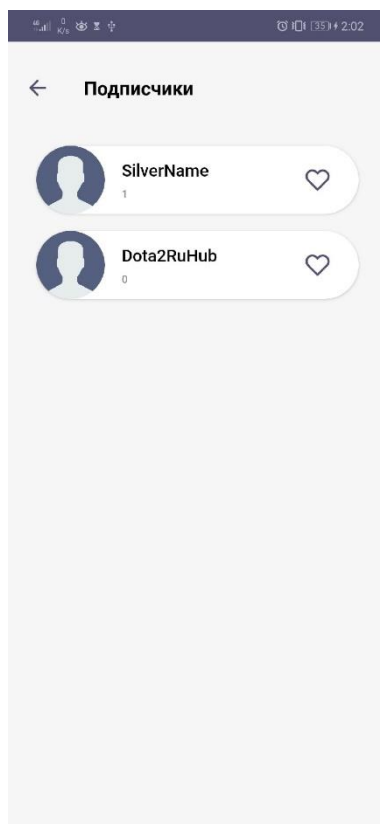


Рисунок Д.8 – Форма «Подписчики» до подписки на другого пользователя

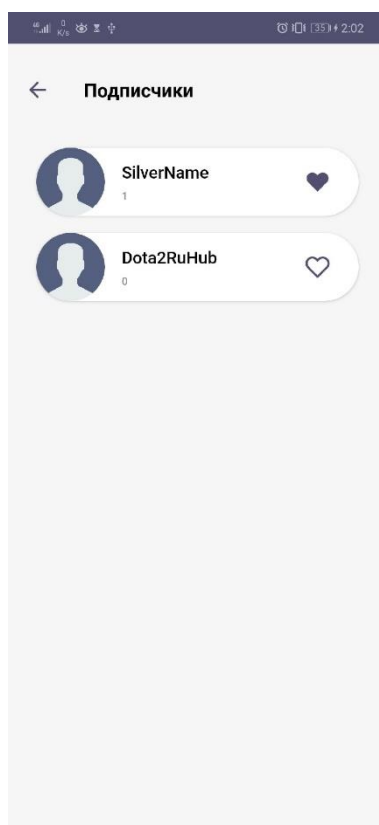


Рисунок Д.9 – Форма «Подписчики» после подписания на другого пользователя

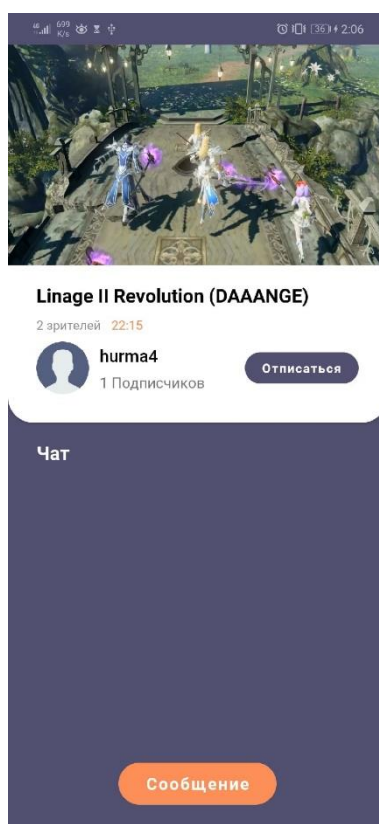


Рисунок Д.10 – Форма «Трансляция» до отписки от стримера

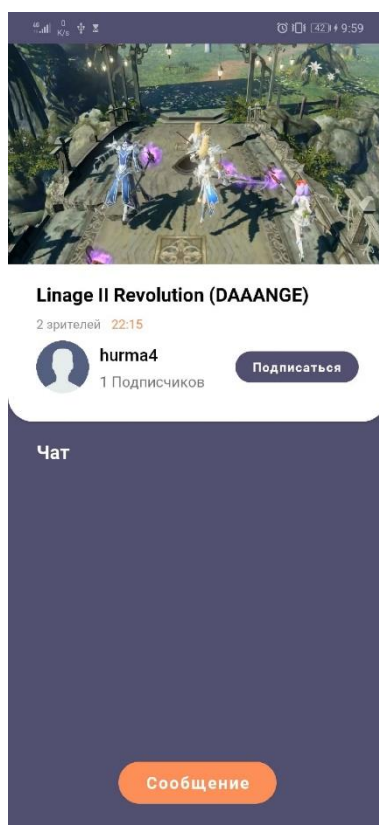


Рисунок Д.11 – Форма «Трансляция» после отписки от стримера

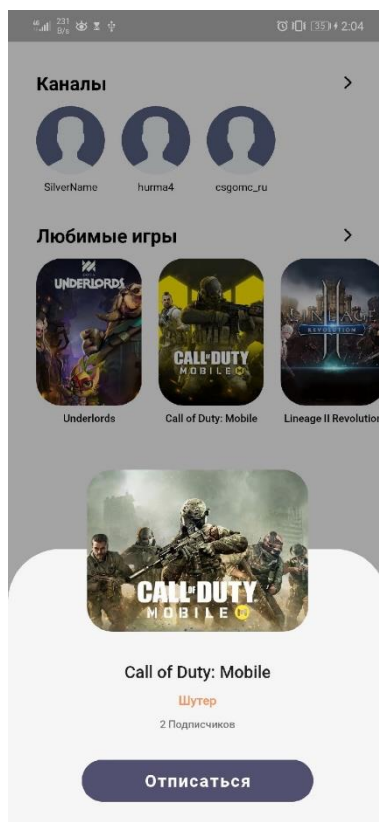


Рисунок Д.12 – Диалог с информацией о игре после подписки

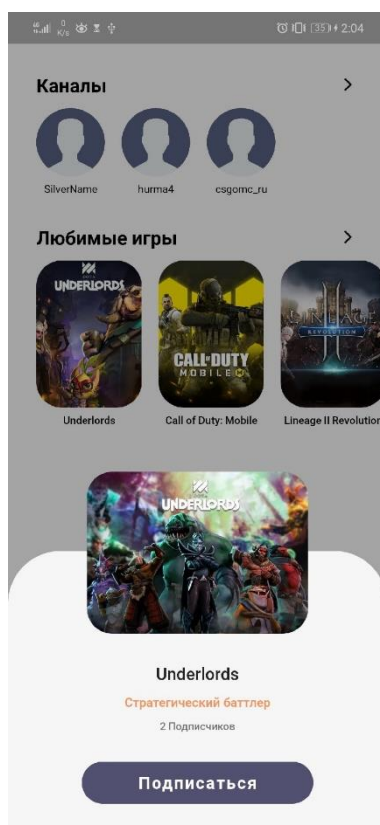


Рисунок Д.13 – Диалог с информацией о игре после отписки

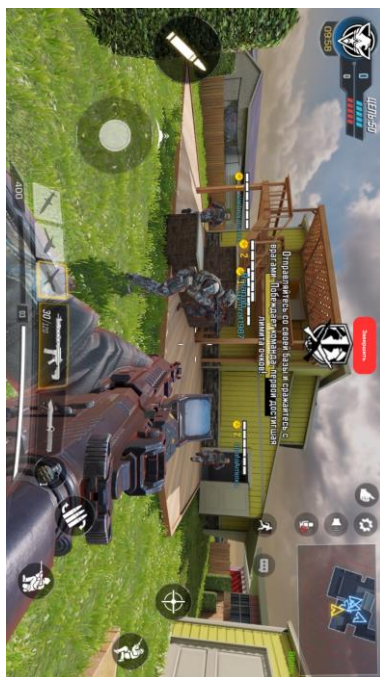


Рисунок Д.14 – Запущенная игра с кнопкой завершения трансляции

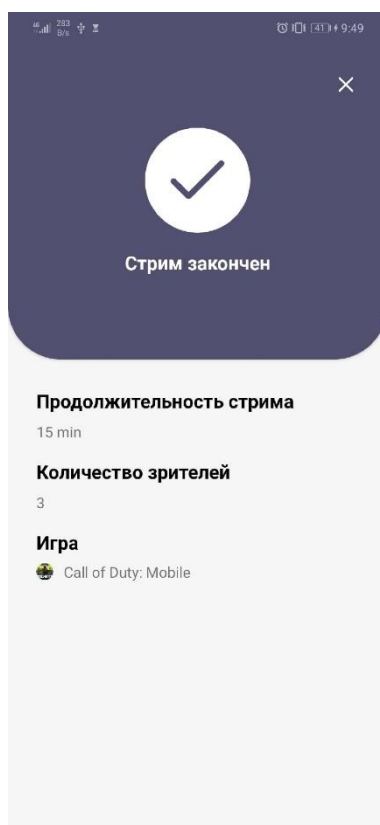


Рисунок Д.15 – Форма с информацией о закончившейся трансляции

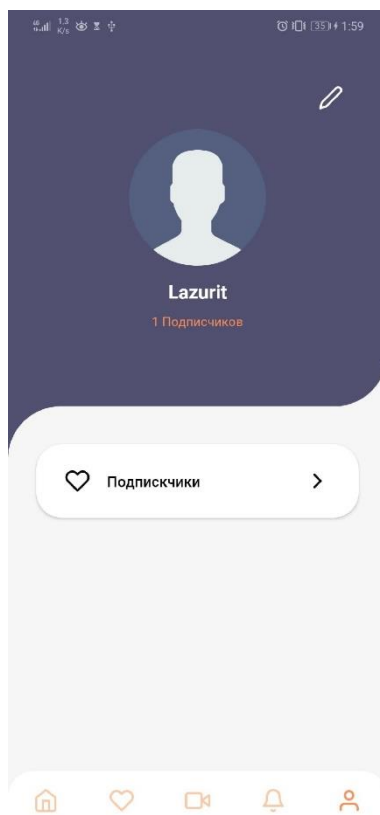


Рисунок Д.16 – Форма «Профиль»

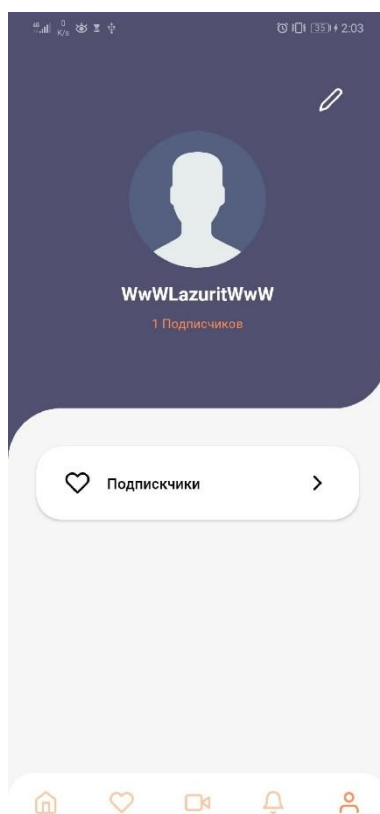


Рисунок Д.17 – Форма «Профиль» после редактирования

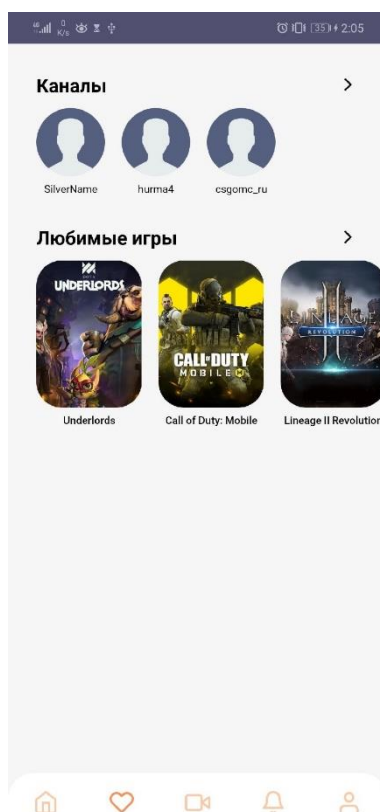


Рисунок Д.18 – Форма «Подписки»

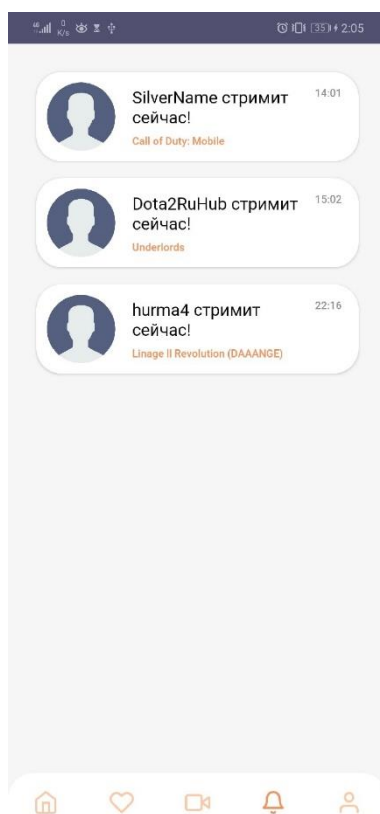


Рисунок Д.19 – Форма «Уведомления»