ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**Отчет о прохождении учебной практики**

|  |
| --- |
| Зотов Максим Сергеевич |

*(Ф.И.О. обучающегося)*

|  |
| --- |
| 2 курс, гр.3530901/90003 |

|  |
| --- |
| 09.03.01 «Информатика и вычислительная техника» |

*(Направление подготовки (код и наименование)*

|  |
| --- |
| **Место прохождения практики**: Высшая школа интеллектуальных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ»  с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
| **Сроки практики:** с 08 июня по 05 июля 2021 г. |

|  |
| --- |
| **Руководитель практики:** |

|  |
| --- |
| Абрамов Н.А. , ассистент ВШИСиСТ ИКНТ |

*(Ф.И.О., уч.степень, должность)*

|  |
| --- |
| **Оценка (зачет): \_\_\_\_\_\_\_\_\_\_\_** |

|  |
| --- |
| Руководитель практики: / Абрамов Н.А. / |

|  |
| --- |
| Обучающийся: / Зотов М.С. / |

|  |
| --- |
| Дата: |

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

***«*САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и технологий

**ИНДИВИДУАЛЬНЫЙ ПЛАН (ЗАДАНИЕ И ГРАФИК)**

**ПРОВЕДЕНИЯ ПРАКТИКИ**

|  |
| --- |
| Ф.И.О. обучающегося Зотов Максим Сергеевич |

|  |
| --- |
| **Направление подготовки** (код/наименование): 09.03.01, «Информатика и вычислительная техника» |
| **Вид практики:** учебная |
| **Тип практики:** ознакомительная |
| **Место прохождения практики**: Высшая школа интеллектуальных систем и суперкомпьютерных технологий (ВШИСиСТ) ИКНТ ФГАОУ ВО «СПбПУ». При необходимости руководителями практики обеспечивается организация практики на базе университета с использованием электронного обучения и дистанционных образовательных технологий. |

|  |
| --- |
|  |

|  |
| --- |
|  |
| Руководитель практики: Абрамов Н.А. , ассистент ВШИСиСТ ИКНТ |
| *(Ф.И.О., уч.степень, должность)* |

**Рабочий график проведения практики**

Сроки практики: с **08.06.2021** г. по **05.07.2021** г.

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Этапы (периоды) практики | Вид работ | Сроки прохождения этапа (периода) практики |
| 1 | Организационный этап | Установочная лекция для разъяснения целей, задач, содержания и порядка прохождения практики, выдача сопроводительных документов по практике  Основная тема: разработка Android-приложения для онлайн викторины | 08.06.2021 г. |
| 2 | Основной  этап | Сбор информации, обработка, систематизация и анализ фактического и теоретического материала.  Содержание практики: (изучение платформы Android, принципов создания клиент-серверных приложений, основ работы с базами данных; разработка Android-приложения, сервера и базы данных)  Планируемые результаты прохождения практики: (Android-приложение для онлайн викторины) | 09.06.2021 - 04.07.2021 г. |
| 3 | Заключительный этап | Подготовка отчета | 01.07.- 04.07.2021 г. |
| Защита отчета по практике (зачет) | 05.07.2021 г |

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Зотов М.С. /

Руководитель практики \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / Абрамов Н.А. /

Оглавление

[1. Техническое задание 4](#_Toc75817460)

[2. Метод решения 5](#_Toc75817461)

[3. Работа программы 20](#_Toc75817462)

[4. Заключение 23](#_Toc75817463)

[5. Источники 23](#_Toc75817464)

# **1. Техническое задание**

Нужно разработать приложение для Android со следующей функциональностью:

1) Имеется база данных (которая находится не на устройстве пользователя, а на каком-нибудь хостинге) с набором разных вопросов и вариантами ответов на них (например, вопрос “В каком году были проведены Олимпийские игры в России”, варианты: 2000, 2014, 2020)

2) Есть само Android-приложение, в котором вы можете выбрать себе имя и найти другого пользователя по его имени и предложить сыграть в игру (либо, соответственно, он может найти вас по вашему имени). В случае, если он соглашается, вам задаётся N случайных вопросов из упомянутой в пункте 1 базы данных. На каждый вопрос даётся ограничение M секунд. То есть принцип игры такой: выпадает i-ый вопрос, и в течение M секунд вы должны выбрать верный вариант. Если выбрали неверный, получили минус балл, ничего не выбрали – ноль баллов, выбрали верный – плюс балл. Выигрывает тот, кто за N вопросов наберёт больше баллов.

3) Существует сервер, который отвечает за обнаружение пользователей, пользующихся в данный момент нашим приложением, за создание игровых сессий для каждой пары игроков, а также за сам процесс игры, то есть именно на сервере определяется, кто дал правильный или неправильный ответ – Android приложение лишь сообщает серверу ответы игроков и отображает информацию о том, оказались ли ответы игроков верны.

Таким образом, архитектуру приложения можно визуализировать примерно следующим образом:

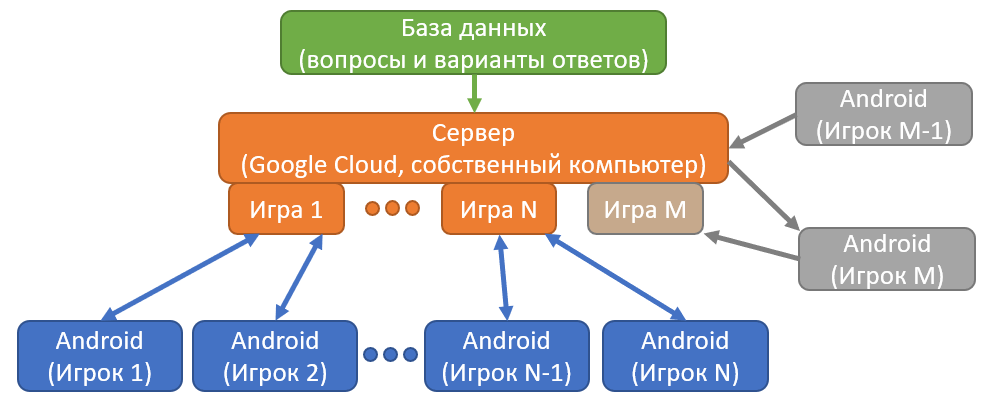


Рис. 1 Архитектура приложения

Из рисунка видно, что сервер получает список вопросов из базы данных. Кроме того, в «Сервер» надпись “Google Cloud, собственный компьютер” указывает некоторые варианты того, где сервер может быть развёрнут (вообще я хочу попробовать в Google Cloud развернуть, но, если вдруг не получится, сделаю на своём ПК). Также рисунок демонстрирует пример создания игровой сессии: игрок M-1 обращается к серверу с запросом найти игрока M, сервер находит игрока M, и, если игрок M соглашается сыграть, создаётся игра M.

Примерный вид приложения:

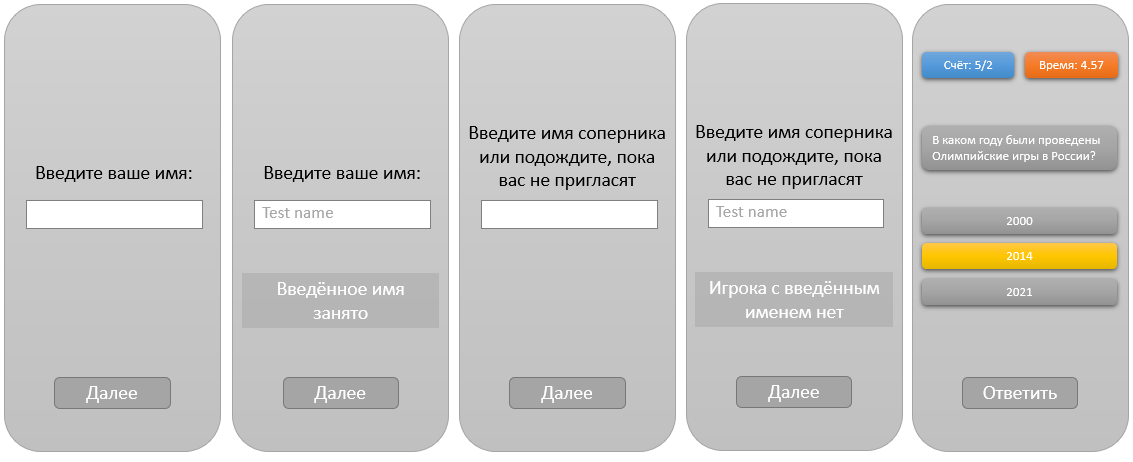


Рис. 2 Пользовательский интерфейс

Репозиторий сервера: <https://github.com/MaksimZotov/quiz-server>

Репозиторий android-приложения: <https://github.com/MaksimZotov/quiz-android>

Ссылка на отчёт: <https://github.com/MaksimZotov/quiz-android/tree/master/documents>

# **2. Метод решения**

**2.1 База данных**

Как и было сказано в ТЗ, вопросы берутся из базы данных. Базу данных я разместил на хостинге <https://sprinthost.ru/> и создал в ней таблицу со следующим содержимым:

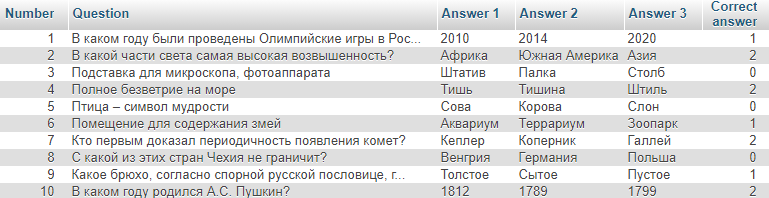


Рис. 3 Вопросы, расположенные в базе данных

Столбец Number – номер вопроса, по этому номеру на стороне сервера я буду запрашивать нужный мне вопрос.

Question – текст вопроса.

Answer 1, 2, 3 – варианты ответов.

Correct Answer – индекс корректного ответа (считаем индекс с нуля)

**2.2 Сервер**

Сервер развёрнут в Google Cloud.

**2.2.1 Классы для обмена информацией между сервером и Android-приложением**

Классы для обмена информацией расположены в пакете data в файле Data.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/data/Data.kt>

Интерфейс Data – его реализуют все классы, предназначенные для общения приложения и сервера, кроме того, этот интерфейс реализует Serializable, чтобы объекты типа Data можно было передавать по сети.

Класс Name(val name: String) – клиент (то есть Android-приложение) отправляет серверу запрос присвоить этому клиенту имя name.

Класс RefusalTheName(val name: String) – сервер отправляет клиенту уведомление о том, что имя name занято.

Класс AcceptingTheName(val name: String) – сервер отправляет клиенту уведомление о том, что имя name назначено этому клиенту.

Класс NameChange() – клиент говорит серверу, что отказывается от своего предыдущего имени, но новое имя он (т.е. клиент) еще не выбрал.

Класс Invitation(val name: String) – клиент сообщает серверу, что хочет пригласить клиента с именем name сыграть в игру.

Класс InvitationMyself() – сервер говорит клиенту, что тот попытался пригласить самого себя.

Класс IncorrectInvitation(val name: String) – сервер уведомляет клиента, что тот пригласил игрока с именем name, однако игрока с таким именем не существует.

Класс InvitedPlayerIsDecidingWhetherToPlayWithAnotherPlayer(val name: String) – сервер сообщает клиенту, что клиент с именем name, которого тот пригласил, в данный момент решает, играть ли с каким-то другим игроком.

Класс ThePlayerWhoInvitedYouIsWaitingForAcceptingTheInvitationFromAnotherPlayer(val name: String) – сервер говорит клиенту, что человек с именем name, ранее пригласивший клиента, в данный момент ждет подтверждение приглашения от другого игрока.

Класс AcceptingTheInvitation(val name: String) – клиент уведомляет сервер, что принимает приглашение на игру от клиента с именем name

Класс IncorrectAcceptingTheInvitation(val name: String) – сервер уведомляет клиента, если тот согласился на приглашение от клиента с именем name, однако клиент с именем name не ждёт соглашение на приглашение (например, клиент name просто закрыл приложение после того, как пригласил нашего клиента).

Класс PlayTheGame() – сервер сообщает двум клиентам, что для них запущена игра.

Класс RefusalTheInvitation(val name: String) – клиент сообщает серверу, что отказывается играть с игроком по имени name / сервер сообщает клиенту, что игрок с именем name отказался от игры.

IncorrectRefusalTheInvitation(val name: String) – сервер говорит клиенту, что тот отверг приглашение от игрока name, в то время как игрок name и не ждал подтверждения приглашения от клиента.

Класс Question(val question: String, val answers: List<String>, val indexOfCorrectAnswer: Int) – сервер отправляет клиенту текущий вопрос для викторины, варианты ответов и индекс корректного ответа.

Класс RemainingTime(val time: Int) – сервер сообщает двум клиентам, что осталось time секунд для ответа на текущий вопрос.

Класс Score(val playerNameToScore: Map<String, Int>) – передаёт мапу, где в качестве ключа имя игрока, в качестве значения – его очки в данный момент игры.

Класс Answer(val indexOfAnswer: Int) – клиент отправляет серверу индекс ответа, который он дал на вопрос.

Класс LeavingTheGame() – клиент сообщает серверу, что покинул игру / сервер сообщает клиенту, что его соперник покинул игру.

Класс FinishTheGame() – сервер сообщает клиентам, что игра завершена.

Класс RequestToPlayAgain() – клиент говорит серверу, что желает еще раз сыграть с текущим соперником.

Класс RefusalToPlayAgain() – клиент говорит, что не хочет заново играть с текущим соперником.

Класс HardRemovalOfThePlayer() – сообщает серверу, что клиента нужно удалить отовсюду / сообщает клиенту, что на стороне его соперника произошла неизвестная ошибка (например, у соперника пропал доступ к сети).

Класс Ping() – с периодичностью в несколько секунд сервер отсылает каждому клиенту экземпляры этого класса.

Класс Pong() – клиент после получения от сервера Ping() в ответ отправляет Pong(). Далее сервер на определённом моменте времени проверяет, все ли клиенты отправили Pong(). Если есть те, кто не отправил, то сервер удаляет их через HardRemovalOfThePlayer(), потому что отсутствие Pong() со стороны клиента трактуется как потеря доступа к интернету у клиента (вообще идею с Ping и Pong пришлось принять на вооружение из-за того, что при использовании сокетов для обмена данными между сервером и клиентом никак нельзя отследить факт потери связи у клиента, а делать это нужно).

**2.2.2 Логирование**

Для того, чтобы была возможность отследить корректность работы сервера, запущенного на удалённой машине, был создан специальный класс Logging – расположен в основном пакете проекта: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/Logging.kt>

Класс Logging содержит свойство tag, которое позволяет выводить в консоль информацию в виде “tag: некоторый текст”. Класс содержит конструктор, в котором можно указать значение tag, а также функцию log(text: String), которая в консоль печатает text.

**2.2.3 Точка входа**

Точкой входа является функция main в файле EntyPoint.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/EntryPoint.kt>

Внутри эта функция обращается к объекту Server и вызывает на нём функцию start().

**2.2.4 Объект Server**

Объект Server расположен в пакете network в файле Server.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/network/Server.kt>

Этот объект (как и некоторые рассматриваемые далее) содержит свойства logging (экземпляр Logging с tag = “Server”) и log (лямбда, вызывающая logging.log(text) – добавил её для лаконичности кода в целом).

Кроме того, этот объект имеет свойство serverSocket типа ServerSocket. serverSocket имеет порт 80, так как именно по этому порту клиенты будут общаться с сервером.

Также Server содержит функцию start(), в которой выводится некоторая информация в log, вызывается функция Cleaner.start() (что она делает, будет сказано позже), а также запускается бесконечный цикл, задача которого – обнаруживать новых клиентов, подключающихся к серверу.

**2.2.5 Объект Cleaner**

Расположен в storage/Cleaner.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/storage/Cleaner.kt>

Cleaner содержит одну функцию start(), которая запускает корутину, внутри которой запускается бесконечный цикл, задача которого в том, чтобы с некоторой периодичностью проверять, что все клиенты отправили Pong(), а также в случае, если кто-то не отправил, сообщать другим классам сервера о необходимости удаления не отправившего Pong() клиента.

**2.2.6 Класс Client**

Класс расположен в пакете network: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/network/Client.kt>

Объекты именно этого класса создаются внутри функции Server.start() в моменты обнаружения новых клиентов. По сути, основная функциональность Client – передавать данные от сервера клиенту и наоборот.

У Client в init {…} запускается корутина с бесконечным циклом внутри. В этом цикле считываются данные, поступающие от клиента

Client имеет функцию sendPong() – она отправляет клиенту Pong().

Также Client содержит свойство receivedPong: Boolean. Когда Cleaner вызывает у Client-а функцию sendPong(), receivedPong становится равным false. Когда в init {…} в корутине в цикле получаем от клиента Pong(), receivedPong становится равным true. Именно так обеспечивается проверка того, отослал ли Client Pong() или нет.

Кроме всего прочего, Client содержит функцию sendDataToClient(data: Data), отправляющие данные клиенту, функцию stop, прекращающую чтение данных от клиента, а также переопределенную для логировния функцию toString().

**2.2.7 Объект ClientsStorage**

Расположен в storage/ClientsStorage.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/storage/ClientsStorage.kt>

Является своего рода хранилищем клиентов, их имен и некоторой информации о клиентах (например, кто в игре, кто просто в онлайне, кто не имеет имени)

**2.2.8 Интерфейс Session**

Расположен в sessions/Session.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/sessions/Session.kt>

Содержит функцию handleDataFromClient(data: Data, client: Client), где data – данные, поступившие от клиента client.

Этот интерфейс реализуют WaitingForNameSession, OnlineSession, GameSession.

**2.2.9 Объект WaitingForNameSession**

Расположен в sessions/WaitingForNameSession.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/sessions/WaitingForNameSession.kt>

Здесь следует дополнить информацию по Client из пункта 2.2.6. У него есть свойство session: Session. Именно session определяет, к какой сессии в данный клиент в данный момент относится. Когда в Server.start() обнаруживается новый клиент и создаётся соответствующий этому клиенту объект класса Client, этому объекту в качествe session указывается WaitingForNameSession, так как у этого клиента еще нет имени.

WaitingForNameSession имеет следующие функции:

handleDataFromClient(data: Data, client: Client) – переопределенная функция, так как WaitingForNameSession реализует интерфейс Session. В функции происходит обработка поступивших от клиентов данных.

handleName(name: Name, client: Client) – из handleDataFromClient() мы идем сюда, если получили от клиента Name(val name: String). Функция проверяет, занято ли имя name, уведомляет клиента об этом, а также перенаправляет клиента в OnlineSession, если имя назначено клиенту.

handleHardRemovalOfThePlayer(client: Client) – полностью удаляет клиента.

addClient(client: Client) – добавляет клиента в сессию WaitingForNameSession.

**2.2.10 Объект OnlineSession**

Расположен в sessions/OnlineSession.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/sessions/OnlineSession.kt>

OnlineSession контролирует, кто кого пригласил сыграть, для каких пар игроков запускать игру, кого отправлять обратно в WaitingForNameSession, если тот захотел изменить своё имя.

Функции:

handleDataFromClient(data: Data, client: Client) – делает то же, что и в WaitingForNameSession.

addClient(client: Client) – добавляет клиента client в OnlineSession

createGameSession(whoInvited: Client, whoIsInvited: Client) – создаёт игровую сессию для двух игроков.

waitForAcceptingTheInvitation(whoInvited: Client, whoIsInvited: Client) – сохраняет информацию о том, что клиент whoInvited пригласил сыграть клиента whoIsInvited.

removeClientFromOnlineSession(client: Client) – удаляет информацию о клиенте из OnlineSession

handleInvitation(invitation: Invitation, client: Client) – обрабатывает ситуацию, когда клиент client пригласил кого-то сыграть.

handleAcceptingTheInvitation(acceptingTheInvitation: AcceptingTheInvitation, client: Client) – обрабатывает ситуацию, когда игрок client согласился сыграть с игроком по имени acceptingTheInvitation.name.

handleRefusalTheInvitation(refusalTheInvitation: RefusalTheInvitation, client: Client) – обрабатывает ситуацию, когда игрок client отказался сыграть с игроком по имени refusalTheInvitation.name.

handleNameChange(client: Client) – функция вызывается, если клиент захотел поменять имя. В таком случае он добавляется в WaitingForNameSession и удаляется из OnlineSession.

handleHardRemovalOfThePlayer(client: Client) – полностью удаляет клиента.

**2.2.11 Класс GameSession**

Расположен в sessions/GameSession.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/sessions/GameSession.kt>

Задачи класса:

1) передавать клиентам информацию о состоянии игры (у кого сколько очков, текущий вопрос, сколько времени осталось для ответа на него и т.д.),

2) Изменять состояние игры на основе данных со стороны клиентов (например, если клиент отправил верный ответ, увеличить количество его очков на 1)

Первая задача решается через реализацию интерфейса GameStateSender (см. пункт 2.2.12), вторая – так же, как и в WaitingForNameSession и в OnlineSession, через реализацию интерфейса Session (то есть через переопределение функции handleDataFromServer())

**2.2.12 Интерфейс GameStateSender**

Расположен в пакете game: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/game/GameStateSender.kt>

Функции как раз этого интерфейса переопределяются в классе GameSession (см. пункт 2.2.11).

Функции:

sendScore(playerNameToScore: Map<String, Int>) – отправляет информацию об очках игроков.

sendQuestion(question: Triple<String, List<String>, Int>) – отправляет текущий вопрос.

sendRemainingTime(remainingTime: Int) – отправляет оставшиеся на текущий ответ секунды.

sendFinish() – сообщает о том, что игра завершена.

**2.2.13 Класс GameState**

Расположен в game/GameState.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/game/GameState.kt>

Класс хранит состояние игры: количество вопросов в игре, количество вопросов в пуле вопросов (в нашем случае в базе данных), время ответа на вопрос, текущий вопрос, варианты ответов на него, очки игроков, их имена. Также в init {…} этот класс содержит корутину, в которой с определённым периодом формируются новые вопросы и отсчитывается текущее время, оставшееся для ответа.

Функции:

stopGame() – прекращает игру

getAnswer(playerName: String, indexOfAnswer: Int) – обрабатывает ситуацию, когда игрок с именем playerName отправил индекс ответа, равный indexOfAnswer. Если indexOfAnswer совпал с правильным, то количество очков игрока playerName увеличивается на 1, иначе – уменьшается.

getNumberOfQuestion() – случайным образом возвращает номер ответа, который нужно взять из базы данных

**2.2.14 Интерфейс QuestionsPool**

Расположен в questions/QuestionsPool.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/questions/QuestionsPool.kt>

Является общим интерфейсом для всех классов, предоставляющих вопросы для викторины. Содержит одну функцию getQuestion(numberOfQuestion: Int): Triple<String, List<String>, Int>, где numberOfQuestion – номер вопроса, возвращаемый объект – Triple (first – текст вопроса, second – список вариантов, third – индекс верного варианта).

**2.2.15 Объект Database**

Расположен в questions/ Database.kt: <https://github.com/MaksimZotov/quiz-server/blob/master/src/main/kotlin/questions/Database.kt>

Реализует интерфейс QuestionsPool (см. пункт 2.2.14), содержит только одну функцию этого же интерфейса. Особенность реализации в том, что вопросы мы получаем из базы данных (см. пункт 2.1).

**2.3 Android-приложение**

В Android-приложении используется паттерн MVVM. В связи с этим проект имеет следующую структуру:

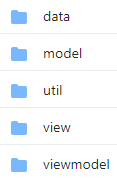


Рис. 4 Структура пакетов в Android-приложении.

Что касается пакета data, то здесь про него можно сказать всё то же самое, что было сказано в пункте 2.2.1. В соответствии с MVVM пакет model хранит в себе состояние приложение (имя игрока, имя противника и т.п.), а также классы для обмена данными между приложением и сервером. Пакет view обеспечивает связку данных и функциональности элементов пользовательского интерфейса с viewmodel, а пакет viewmodel содержит эти данные и функциональность, а также определяет то, какие изменения нужно вносить в model в ответ на те или иные действия пользователя, и осведомляет view (то есть пользовательский интерфейс) о данных, полученных от model. Пакет util содержит разного рода вспомогательные классы.

**2.3.1 Абстрактный класс ConnectionLostCallback**

Расположение: util/ConnectionLostCallback: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/util/ConnectionLostCallback.kt>

Задача класса – уведомлять пользователя о том, что у него пропал доступ к интернету.

Функции:

onLost() – вызывается в случае потери связи

notifyThatConnectionLost() – исполняется в случае потери связи (конкретная реализация не определена, так как эта функция абстрактная).

**2.3.2 Файл Toasts.kt**

Файл находится в util/Toasts.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/util/Toasts.kt>

Файл содержит лишь одну функцию: shortToast(context: Context?, text: String). Эта функция служит оберткой над стандартной функцией вызова уведомления, используется лишь для лаконичности кода.

**2.3.3 Класс SingleLiveData**

Класс находится в util/SingleLiveData.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/util/SingleLiveData.kt>

Данный класс реализован для обхода бага, связанного с кнопкой «назад» при использовании в проекте android navigation component. Данный класс был взят из интернета. Подробнее об этом баге и реализации этого класса можно прочитать по ссылке: <https://stackoverflow.com/questions/59834398/android-navigation-component-back-button-not-working>

**2.3.4 Класс AppState**

Класс находится в model/appstate/AppState.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/appstate/AppState.kt>

Содержит свойства, отражающее состояние приложения (например, имя игрока, имя противника, их очки, ожидает ли текущий игрок подтверждения своего имени со стороны сервера и т.д.)

**2.3.5 Файл DataObserver.kt**

Расположение: model/communication/DataObserver.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/communication/DataObserver.kt>

В файле 2 интерфейса: DataObserver и DataObservable.

DataObserver содержит функции:

subscribeOnObservableData() – в реализации этой функции необходимо будет указывать, какой ViewModel нужно подписываться на данные, поступающие от DataObservable.

getData(data: Data) – в этой функции нужно будет указывать, что делать в случае получения от DataObservable данных data.

DataObservable содержит одну функцию:

setObserver(dataObserver: DataObserver) – в этой функции нудно будет указывать, кого назначить слушателем DataObservable

**2.3.6 Объект ReceiverFromServer**

Расположение: model/communication/ReceiverFromServer.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/communication/ReceiverFromServer.kt>

Класс реализует DataObservable и, соответственно, содержит функцию setObserver(dataObserver: DataObserver) (см. 2.3.5). Кроме того, здесь еще есть функция getData(data: Data), которая будет вызываться извне “читателем” данных с сервера. Внутри этой функции через handler будет вызываться метод getData(data: Data) у текущего DataObserver (handler нужен по причине того, что “читатель” данных с сервера работает не в основном потоке, а информацию о полученных данных надо передать в основной поток).

**2.3.7 Объект SenderToServer**

Расположение: model/communication/SenderToServer.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/communication/SenderToServer.kt>

Задача класса – отправка данных серверу от приложения.

Функции:

sendData(data: Data) – отправляет данные серверу. Стоит отметить, что содержимое функции лежит внутри корутины, так как sendData() будет вызываться из основного потока, а задачи, связанные с работой по сети, требуется выполнять параллельно.

closeConnection() – закрывает соединение с сервером. Делается это тоже внутри корутины.

**2.3.8 Класс Reader**

Расположение: model/network/Reader.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/network/Reader.kt>

Reader считывает данные с сервера. Является наследником класса Thread(). В init {…} вызывается start(), в переопределённом run() в бесконечном цикле происходит считывание данных со стороны сервера.

**2.3.9 Класс Sender**

Расположение: model/network/Sender.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/network/Sender.kt>

Sender отправляет данные серверу. Является наследником класса Thread(). В init {…} вызывается start(), в переопределённом run() выполняется отправка на сервер свойства data, а в функции sendDataToServer(data: Data) свойству data присваивается значение параметра и вызывается run() (для работы параллельно)

**2.3.10 Класс Server**

Расположение: model/network/Server.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/model/network/Server.kt>

Server объединяет в себе задачи создания и закрытия связи с сервером, а также задачи отправки и получения данных.

Функции:

createConnection() – открывает соединение с сервером.

closeConnection() – закрывает соединение.

sendDataToServer(data: Data) – отправляет данные через Sender (2.3.9)

handleDataFromServer(data: Data) – эта функция вызывается объектом класса Reader (2.3.8) и внутри нее data передаётся в ReceiverFromServer.getData(data) (2.3.6).

**2.3.11 Точка входа MainActivity**

Расположена во view/mainactivity/MainActivity.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/view/mainactivity/MainActivity.kt>

В переопределённом методе onCreate() вызывается setContentView(R.layout.activity\_main), и, так как в проекте используется android navigation component, а в res/layouts/activity\_main.xml указан граф навигации, первым, что увидит пользователь после открытия приложения, будет фрагмент, указанный в качестве стартового в этом графе навигации (в нашем случае это будет фрагмент, в котором у пользователя будет спрашиваться его имя).

**2.3.11 Абстрактный класс BaseFragment**

Расположен во view/base/BaseFragment.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/view/base/BaseFragment.kt>

Является базовым для всех остальных фрагментов в приложении. Причиной создания данного класса послужило то, что практически во всех фрагментах приложения требуется выполнение 3 задач:

1) Установление связи между элементами UI и ViewModel-ями через компонент Android, называющийся DataBinding

2) Выполнение тех или иных действий с компонентами UI на основе данных со стороны сервера

3) Переопределение функции, исполняющейся при нажатии на кнопку «назад».

В этой связи в данном классе были добавлены соответствующие 2 абстрактные функции assignBinding() и handleData(data: Data), а также open-функция onBackPressed(). Собственно, задача остальных функций класса главным образом состоит в том, чтобы организовать вызов assignBinding() и handleData() таким образом, чтобы в классах наследниках достаточно было бы просто переопределить их и не беспокоится ни о lifecycle фрагмента, ни о подписке соответствующей данному фрагменту ViewModel-и на ReceiverFromServer (см. пункт 2.3.6).

Кроме того, в данном классе в свойстве connectivityCallback содержится обработка события отключения от интернета.

**2.3.12 Классы AuthenticationFragment, SearchOnNameFragment, InvitationToPlayFragment, GameFragment, FinishGameFragment**

Классы расположены в пакете view: <https://github.com/MaksimZotov/quiz-android/tree/master/app/src/main/java/com/maksimzotov/quiz/view>

У всех этих классов общая структура:

assignBinding() – устанавливает связь между элементами интерфейса и их функциональностью

handleData(data: Binding) – вносит изменения в UI на основе данных от сервера.

onBackPressed() – описывает, что необходимо сделать в случае нажатия на кнопку «назад».

**2.3.13 Класс BaseViewModel**

Расположен во viewmodel/base/BaseViewModel.kt: <https://github.com/MaksimZotov/quiz-android/blob/master/app/src/main/java/com/maksimzotov/quiz/viewmodel/base/BaseViewModel.kt>

Класс является родительским для других ViewModel-ей (сам наследует ViewModel и реализует интерфейс DataObserver (2.3.5))

Содержит свойство data типа SingleLiveData – идея здесь в том, чтобы при получении данных от сервера соответствующим образом изменять data, тем самым уведомляя соответствующий фрагмент из пакета view об изменении (так как SingleLiveData наследуется от MutableLiveData).

Есть функции:

closeConnection() – закрывает соединение с сервером

subscribeOnObservableData() – подписывает данный экземпляр на ReceiverFromServer.

getData(data: Data) – ReceiverFromServer вызывает эту функцию при получении данных с сервера, а сама эта функция изменяет значение свойства data, тем самым уведомляя соответствующий фрагмент о поступлении данных от сервера.

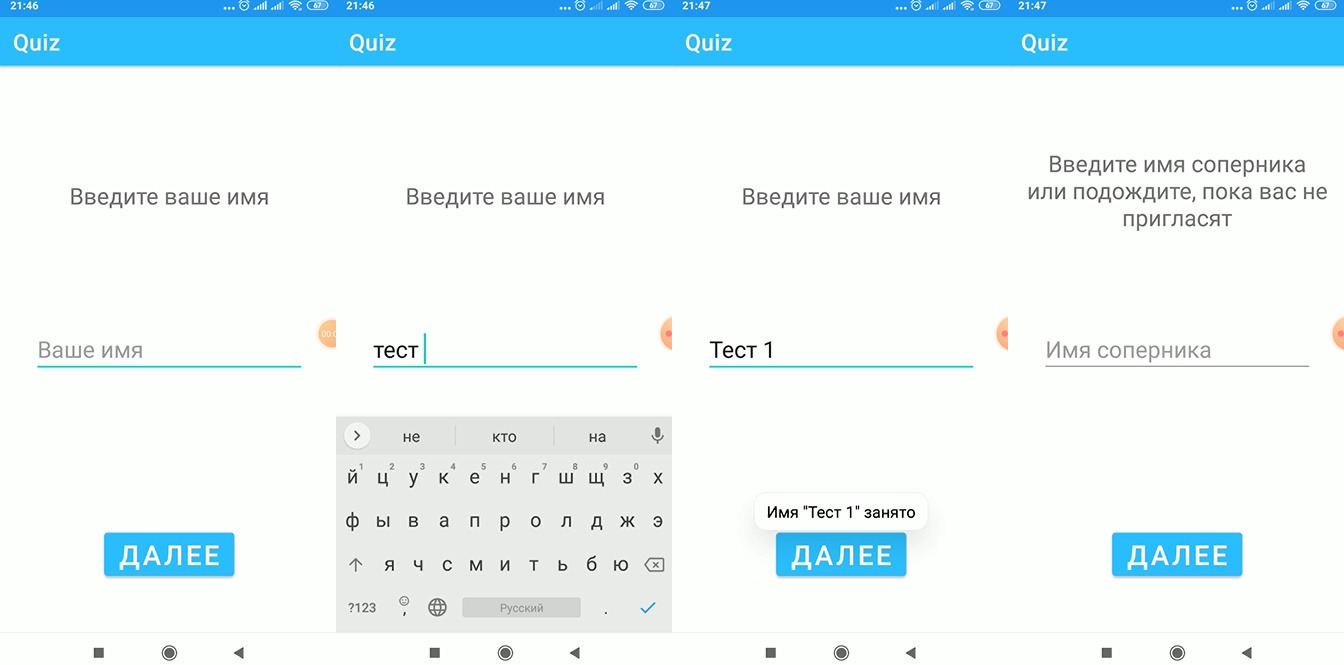
**2.3.14 Классы AuthenticationViewModel, SearchOnNameViewModel, InvitationToPlayViewModel, GameViewModel, FinishGameViewModel**

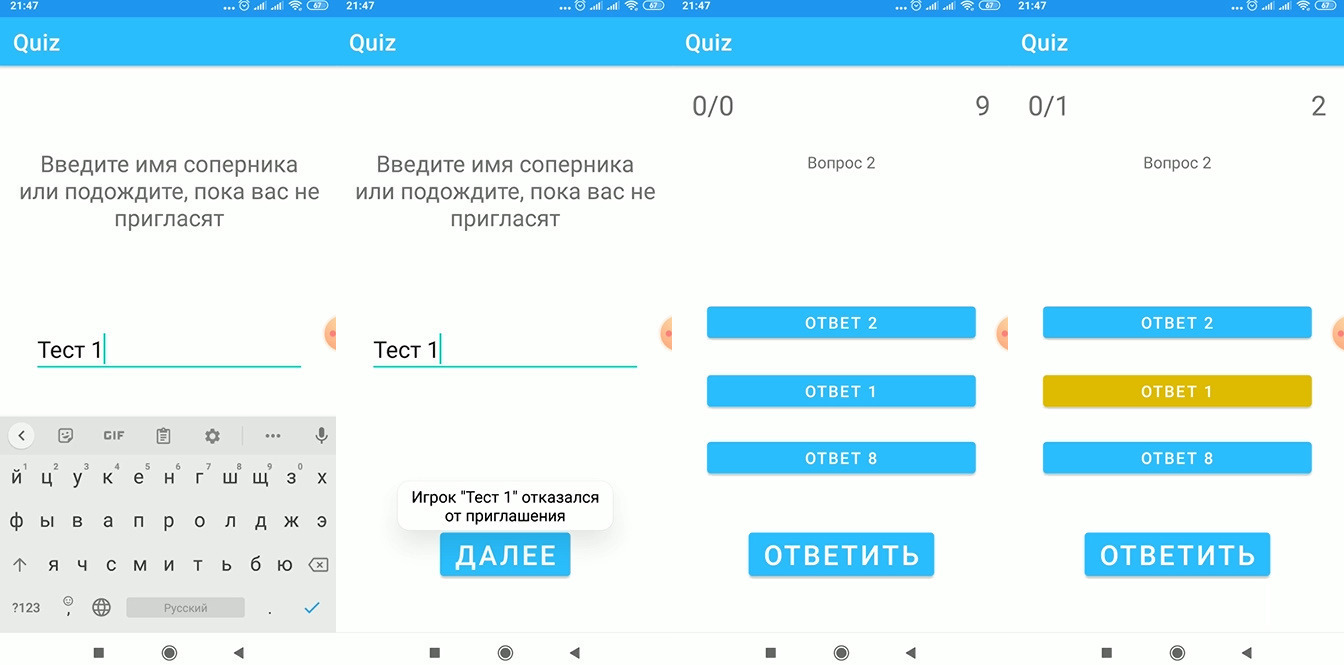
Расположение: пакет viewmodel: <https://github.com/MaksimZotov/quiz-android/tree/master/app/src/main/java/com/maksimzotov/quiz/viewmodel>

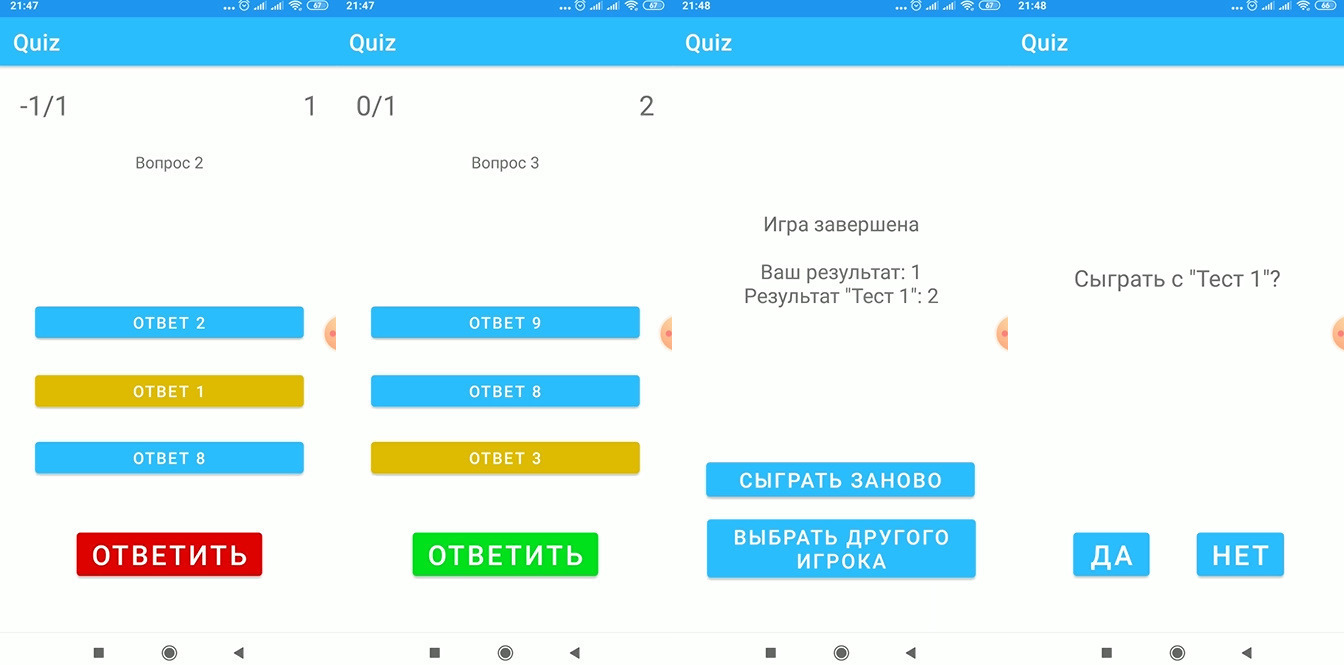
Все данные ViewModel-и наследуют BaseViewModel и отражают функциональность и данные, соответствующие их фрагментам. Эти классы содержат крайне короткие функции, названия которых полностью описывают, что они делают, поэтому, если и описывать эти функции, то их описание сведётся к переводу названий функций на русский язык, что, по сути, не имеет никакого смысла.

# **3. Работа программы**

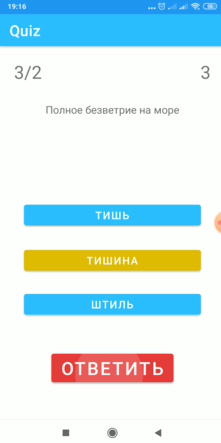
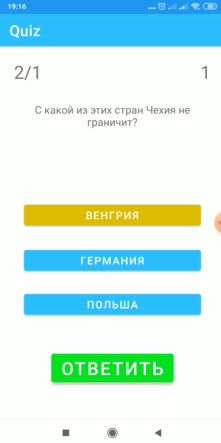
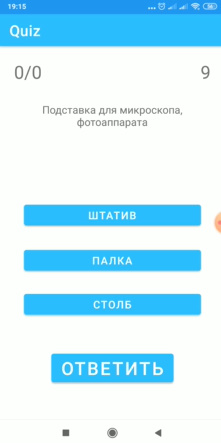
Скриншоты работы программы взяты из видео работы программы – по этой причине справа есть значок, уведомляющий о том, что идёт запись. Разумеется, в самом приложении этого значка нет.







**Примеры реальных вопросов:**



# **4. Заключение**

В ходе работы был получен опыт работы с базой данных, платформой Google Cloud (где был развёрнут сервер), а также был приобретён опыт разработки под Android. Я закрепил навыки работы с сокетами, познакомился с такими инструментами Android, как DataBinding, LiveData, Handler и другими. По итогам работы были выполнены все задачи, поставленные в техническом задании.

# **5. Источники**

<https://developer.android.com/docs> - документация по Android от Google

<https://cloud.google.com/docs/tutorials?hl=ru> – гайды по работе с Google Cloud

<https://www.fandroid.info/> - сайт по android-разработке

<http://developer.alexanderklimov.ru/android/index.php> - ещё один сайт по android-разработке