Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика, искусственный интеллект и системы управления (ИУ)»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии (ИУ7)»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Онлайн-игра «ПОКЕР»

Студент ИУ7-32М _____ В. А. Филипенков
(Группа) (Подпись, дата) (И. О. Фамилия)

Студент ИУ7-32М _____ А. Е. Лахов
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. М. Никульшин
(Подпись, дата) (И. О. Фамилия)

2025 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
(Индекс)
_____ И. В. Рудаков
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине Протоколы вычислительных сетей

Студенты группы ИУ7-32М

Филипенков Владислав Александрович
(Фамилия, имя, отчество)
Лахов Александр Евгеньевич
(Фамилия, имя, отчество)

Тема курсового проекта Онлайн-игра «ПОКЕР»

Направленность КП (учебный, исследовательский, практический, производственный, др.)
учебный
Источник тематики (кафедра, предприятие, НИР) Кафедра

График выполнения проекта: 25% к 4 нед., 50% к 7 нед., 75% к 11 нед., 100% к 14 нед.

Задание Разработать протокол для онлайн-игры «ПОКЕР». Проанализировать предметную область, выделить целевую аудиторию и сценарии его применения. Определить участников игрового процесса и их функции. Определить способы их взаимодействия и составить набор соответствующих сообщений протокола и их содержание. Протокол должен поддерживать проверку целостности данных, повторный запрос потерянных и повреждённых сообщений. Реализовать и протестировать клиент-серверное приложение, использующее данный протокол.

Оформление курсового проекта:

Расчетно-пояснительная записка на 15-30 листах формата А4.
Расчетно-пояснительная записка должна содержать постановку задачи, введение, аналитическую, конструкторскую, технологическую части, заключение и список литературы.

Дата выдачи задания «16» октября 2024 г.

Руководитель курсового проекта _____ А.М.Никульшин
(Подпись, дата) (И.О.Фамилия)
Студент _____ В.А. Филипенков
(Подпись, дата) (И.О.Фамилия)
Студент _____ А.Е. Лахов
(Подпись, дата) (И.О.Фамилия)

СОДЕРЖАНИЕ

РЕФЕРАТ

Курсовая работа *«Протокол для онлайн-игры «ПОКЕР»*.

В рамках данной работы был разработан протокол для онлан-игры в покер по правилам Техасского Холдема, работающий поверх протоколов передачи данных HTTP, WebSocket и протокола аутентификации – OpenId Connect.

Также было разработано клиент-серверное приложение с графическим пользовательским интерфейсом, в полной мере реализующее разработанный протокол.

Расчетно-пояснительная записка содержит 62 страницы, 19 рисунков, 29 таблиц, 15 источников, 1 приложение.

ВВЕДЕНИЕ

ПОКЕР (англ. poker) – одна из самых популярных в мире карточных игр и одна из немногих карточных игр, по которой проводятся соревнования мирового уровня. Играют колодой в 52 карты; цель игрока – получить наиболее выигрышную комбинацию на пяти картах [1].

С развитием индустрии онлайн развлечений и игр встает вопрос о необходимости реализации покера онлайн. Актуальность разработки протокола для онлайн-игры «ПОКЕР» обусловлена необходимостью надежной и эффективной передачи большого потока данных в режиме реального времени в рамках текущей партии для обеспечения комфортного процесса игры для каждого из игроков [2].

Цель курсовой работы – разработать протокол для онлайн-игры «ПОКЕР».

Для достижения поставленной цели необходимо решить следующие задачи:

— провести обзор предметной области;

— выделить целевую аудиторию;

— определить участников игрового процесса и их функции;

— определить способы взаимодействия игроков в рамках партии и составить набор соответствующих сообщений протокола и их содержание;

— реализовать и протестировать клиент-серверное приложение, реализующее данный протокол.

1 Аналитическая часть

1.1 Обзор предметной области

1.1.1 Колода и комбинации

Как правило, все разновидности покера играются с использованием французской колоды в 52 карты (без Джокеров). По старшинству карты располагаются следующим образом: Туз – *Ace* (A), Король – *King* (K), Дама – *Queen* (Q), Валет – *Jack* (J), 10, 9, 8, 7, 6, 5, 4, 3, 2.

Основная цель – получить наиболее выигрышную комбинацию на пяти картах. Иерархия комбинаций покера представлены ниже, в таблице [3].

Таблица 1.1: Иерархия комбинаций в покере

Комбинация	**Кол-во комбинаций**	**Описание**
1. Роял-Флеш *(Royal Flush)*	4	5 старших карт одной масти
2. Стрит-Флеш *(Straight Flush)*	36	5 карт подряд одной масти
3. Каре *(Four of a kind)*	624	4 карты одного ранга
4. Фулл-Хаус *(Full House)*	3 744	3 карты одного ранга и 2 другого
5. Флеш *(Flush)*	5 108	5 карт одной масти
6. Стрит *(Straight)*	10 200	5 последовательных карт
7. Тройка *(Three of a kind)*	54 912	3 карты одного ранга
8. Две пары *(Two pairs)*	123 552	2 карты одного ранга и 2 другого
9. Пара *(Pair)*	1 098 240	2 карты одного ранга
10. Старшая карта *(High card)*	1 302 540	при отсутствии других комбинаций

1.1.2 Классический покер

Правила игры в классический покер следующие [4]:

1. Игра начинается с раздачи каждому игроку по 5 карт лицом вниз.
2. После раздачи карт начинается первый раунд торговли. Игроки имеют возможность делать ставки, поднимать ставки, сбрасывать карты или оставаться в игре. Торговля происходит по часовой стрелке вокруг стола, начиная с игрока, сидящего слева от дилера.
3. По окончании первого раунда торговли игроки могут запросить до трех новых карт, сбросив старые. Запрос новых карт называется «обмен». Каждый игрок выбирает, сколько карт он хочет сбросить и получить новые взамен.
4. После обмена происходит второй раунд торговли. Игроки снова могут делать и поднимать ставки, сбрасывать карты или поддерживать игру без внесения фишек в банк.
5. После второго раунда торговли следует открытие карт (шоудаун). Игроки, оставшиеся в игре, показывают карты, и тот, у кого самая сильная комбинация, выигрывает банк.

Победитель раунда определяется по самой сильной комбинации. Если на столе есть одинаковые комбинации, победитель определяется по старшей карте в комбинации и т.д. Собранный банк за все раунды уходит победителю.

Также существует множество разновидностей покера, каждая из которых имеет уникальные правила и особенности. Некоторые из самых популярных разновидностей включают:

— Техасский Холдем;

— Омаха;

— Семикарточный Стад;

— Разз покер.

1.1.3 Виды покера

Техасский Холдем

Техасский Холдем – одна из самых популярных и широко распространенных разновидностей покера.

Основные правила игры в Техасский Холдем следующие [4]:

1. Начало игры. Игра начинается с раздачи каждому игроку по две закрытые карты, которые называются «карманными картами» или «картами на руках». Эти карты видят только сами игроки.

2. Первый раунд торговли (Префлоп). Игрок, сидящий слева от дилера (или слепого), начинает действие. Игроки могут сделать одно из следующих действий: сделать ставку (бет *BET*), сбросить карты (фолд *FOLD*), поднять ставку (рейз *RAISE*), или просто пасовать (карты при этом останутся на руках) (чек *CHECK*). Торговля происходит по часовой стрелке вокруг стола.

3. Флоп. На столе открываются три общие карты лицом вверх. Этот набор карт называется «флоп». Игроки могут использовать открытые карты в комбинации со своими карманными для составления пятикарточного комбо.

4. Второй раунд торговли (Флоп-торговля). Следующий раунд торговли начинается с игрока, сидящего слева от дилера. Игроки снова могут делать и поднимать ставки, сбрасывать карты или пасовать.

5. Терн. На столе открывается четвертая общая карта лицом вверх. Эта карта называется «терн» или «четвертой улицей».

6. Третий раунд торговли (Терн-торговля). Игроки снова имеют возможность сделать ход, по аналогии с предыдущими торговыми улицами.

7. Ривер. Открывается пятая и последняя общая карта лицом вверх. Эта карта называется «ривер» или «пятой улицей».

8. Финальный раунд торговли (Ривер-торговля). Игроки делают ставки и пытаются определить, кто имеет самую сильную комбинацию.

9. Шоудаун. Если после последнего раунда торговли остаются два или более игроков, происходит шоудаун. Игроки открывают свои карты, и победитель определяется по самой сильной пятикарточной комбинации, составленной из двух карманных и пяти открытых общих карт. Если у игроков комбинации равные, банк делится между ними.

Омаха

Омаха — популярная разновидность покера с отличиями от Техасского Холдема. Игроки получают четыре закрытые карты вместо двух. Главные особенности Омахи [4]:

— игрокам раздают четыре закрытые карты и пять общих карт на столе;

— цель – составить наилучшую пятикарточную комбинацию, используя две из своих карт и три общих карты;

— из-за большего количества карт, комбинации в Омахе обычно более сильные;

— обычно играется в формате Пот-Лимит, где размер ставок ограничен текущим размером банка;

— игроки имеют больше вариантов для составления комбинаций, что требует тщательного анализа и стратегии.

Игра в Омаху требует более сложных расчетов и анализа.

Семикарточный Стад

Семикарточный Стад – разновидность покера, где каждому игроку раздаются семь карт, но только две из них остаются закрытыми. Разновидность выделяется такими аспектами [4]:

— игроки получают три закрытые и четыре открытые карты;

— игроки делают ставки и принимают решения после каждой открытой карты;

— при определении победителя учитывается старшая карта в комбинации, а затем — комбинации карт;

— игроки видят только свои закрытые карты и открытые карты других игроков;

— устанавливается ограничение на количество ставок в каждом раунде;

— цель – составить наилучшую пятикарточную комбинацию из семи карт;

— игроки имеют больше информации о руках других игроков, что требует глубокого анализа и стратегии.

Разз покер

Разз – разновидность покера, где целью является составление самой слабой пятикарточной комбинации. Особенности Разза [4]:

— игрокам раздаются семь карт, две закрытые и пять открытых;

— цель игры – составить самую слабую пятикарточную комбинацию;

— стрит или флэш не учитываются, и чем ниже комбинация, тем сильнее она считается;

— игроки делают ставки или пасуют после каждого раунда раздачи карт. Торговля продолжается до тех пор, пока у игроков остаются карты на руках;

— старшая карта имеет наименьшую ценность;

— банк делится между игроками с самыми слабыми руками.

В данной курсовой работе предлагается разработать протокол для игры в покер по правилам Техасского Холдема, т.к. это один из самых распространенных видов покера и наиболее удобен для игры онлайн.

1.2 Протоколы передачи данных

HTTP

HTTP (англ. HyperText Transfer Protocol – «протокол передачи гипертекста») – протокол прикладного уровня передачи данных, изначально – в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных. Основным объектом манипуляции в HTTP является ресурс, на который указывает URI (Uniform Resource Identifier) в запросе клиента. Обмен сообщениями идёт по обыкновенной схеме «запрос-ответ». Для идентификации ресурсов HTTP использует глобальные URI. В отличие от многих других протоколов, HTTP не сохраняет своего состояния. Это означает отсутствие сохранения промежуточного состояния между парами «запрос-ответ».

HTTP/1.1 принес множество улучшений по сравнению с предыдущими версиями (поддержка постоянных соединений, кэширование, частичные запросы, управление ошибками)

Несмотря на все улучшения, HTTP/1.1 имеет свои ограничения.

Ограничение на количество одновременных соединений: Браузеры ограничивают количество одновременных соединений к одному домену, что может замедлить загрузку страниц. Обычно это ограничение составляет 6-8 соединений на домен, что может быть недостаточно для современных веб-страниц с множеством ресурсов.

Заголовок блокировки (Head-of-line blocking): Если один запрос задерживается, это может заблокировать все последующие запросы в очереди. Это приводит к тому, что время загрузки страницы увеличивается, особенно если один из запросов требует значительного времени на обработку.

Высокая латентность: Из-за необходимости устанавливать новое соединение для каждого ресурса, время загрузки страниц может быть высоким. Даже с поддержкой постоянных соединений, каждый запрос требует отдельного цикла запрос-ответ, что увеличивает задержку.

HTTP/2 (изначально HTTP/2.0) – вторая крупная версия сетевого протокола HTTP, основан на SPDY. Протокол HTTP/2 является бинарным. По сравнению с предыдущим стандартом изменены способы разбиения данных на фрагменты и транспортирования их между сервером и клиентом. В HTTP/2 сервер имеет право послать то содержимое, которое ещё не было запрошено клиентом. Это позволит серверу сразу выслать дополнительные файлы, которые потребуются браузеру для отображения страниц, без необходимости анализа браузером основной страницы и запрашивания необходимых дополнений.

FTP

FTP (англ. File Transfer Protocol) — протокол передачи файлов по сети, появившийся в 1971 году задолго до HTTP и даже до TCP/IP, благодаря чему является одним из старейших прикладных протоколов. Изначально FTP работал поверх протокола NCP[1], на сегодняшний день широко используется для распространения ПО и доступа к удалённым хостам. В отличие от TFTP, гарантирует передачу (либо выдачу ошибки) за счёт применения квотируемого протокола. Протокол построен на архитектуре «клиент-сервер» и использует разные сетевые соединения для передачи команд и данных между клиентом и сервером. Пользователи FTP могут пройти аутентификацию, передавая логин и пароль открытым текстом, или же, если это разрешено на сервере, они могут подключиться анонимно. Поддерживает двустороннее соединение и имеет встроенную аутентификацию пользователей. FTP-аутентификация использует схему имя пользователя/пароль для предоставления доступа. Имя пользователя посылается серверу командой USER, а пароль — командой PASS. Если предоставленная клиентом информация принята сервером, то сервер отправит клиенту приглашение и начинается сессия.

RTP

Протокол RTP (англ. Real-time Transport Protocol) работает на прикладном уровне и используется при передаче трафика реального времени. Протокол RTP переносит в своём заголовке данные, необходимые для восстановления аудио данных или видеоизображения в приёмном узле, а также данные о типе кодирования информации (JPEG, MPEG и т. п.). В заголовке данного

протокола, в частности, передаются временная метка и номер пакета. Эти параметры позволяют при минимальных задержках определить порядок и момент декодирования каждого пакета, а также интерполировать потерянные пакеты. Информация, предоставляемая посредством этого протокола, включает в себя отметку времени (для синхронизации), последовательный номер (для детектирования потери и дублирования пакетов) и формат полезной нагрузки, который определяет формат кодирования данных.

WebSocket

WebSocket — протокол связи поверх TCP-соединения для обмена сообщениями между клиентом и сервером, он использует постоянное соединение. Соединение WebSocket начинается с WebSocket Handshake, который является специальным начальным HTTP-запросом, инициируемым клиентом. Этот запрос содержит особые заголовки, указывающие, что клиент хочет переключиться на WebSocket-протокол. После установки соединения WebSocket данные передаются в виде фреймов — компактных порций данных, которые могут быть отправлены в любом направлении. Фреймы WebSocket минимальны по размеру, что позволяет сократить объем передаваемых данных и уменьшить задержку. WebSockets подходят для сценариев, когда требуется двусторонняя связь между клиентом и сервером. Это идеальный выбор, если приложение должно отправлять данные как от клиента к серверу, так и от сервера к клиенту в режиме реального времени.

SSE

SSE (Server-Sent Events) — протокол для обмена данными между сервером и клиентом в реальном времени, позволяющая клиенту получать автоматические обновления с сервера через HTTP-соединения. Не поддерживает двойное соединение. Принцип работы: клиент подписывается на события сервера, и как только происходит событие, клиент сразу получает уведомление и некоторые данные, связанные с этим событием. Подходит для приложений, где сервер должен отправлять данные клиенту, но не требуется ответных сообщений от клиента в реальном времени.

1.2.1 Протоколы аутентификации пользователей

SAML

SAML — это стандарт для обмена аутентификационными и авторизационными данными между различными доменами. Он часто используется в корпоративных средах для единого входа (SSO). SAML позволяет пользователям аутентифицироваться один раз и получать доступ к различным системам и приложениям без повторного ввода учетных данных. Это значительно упрощает управление доступом и повышает безопасность, так как уменьшает количество точек входа для потенциальных атак.

Kerberos

Kerberos — это сетевой протокол аутентификации, который работает на основе билетов. Он обеспечивает безопасную передачу данных в незащищенных сетях, таких как интернет. Kerberos широко используется в корпоративных сетях и операционных системах, таких как Windows и Unix. Основное преимущество Kerberos заключается в его высокой безопасности и поддержке единого входа (SSO), что делает его идеальным выбором для крупных организаций с высокими требованиями к безопасности.

OAuth 2.0

OAuth 2.0 — это протокол авторизации, который позволяет приложениям получать ограниченный доступ к пользовательским данным без передачи пароля. Он широко используется в веб-приложениях и мобильных приложениях. OAuth 2.0 предоставляет гибкость и масштабируемость, что делает его идеальным для использования в крупных системах и облачных сервисах. Протокол поддерживает различные типы грантов, такие как авторизационный код, имплицитный грант, пароль и клиентские учетные данные, что позволяет адаптировать его под разные сценарии использования

OIDC

OpenID Connect (OIDC) – протокол проверки подлинности удостоверений, который является расширением открытой проверки подлинности (OAuth) 2.0 и предназначен для стандартизации процесса проверки подлинности и авторизации пользователей при входе в систему для доступа к цифровым службам. OIDC обеспечивает проверку подлинности, что означает подтверждение того, что пользователи являются теми, кем они себя называют.

OpenID Connect обеспечивает удобство для пользователей, позволяя им аутентифицироваться с помощью уже существующих учетных записей в популярных сервисах, таких как Google или Facebook. Это упрощает процесс регистрации и входа в систему, улучшая пользовательский опыт.

Таблица 1.2: Сравнение протоколов аутентификации пользователей

Протокол	**Преимущества**	**Недостатки**
OAuth 2.0	Безопасность, гибкость, масштабируемость	Сложность реализации, требует управления токенами
OpenID Connect	Удобство для пользователей, безопасность	Зависимость от сторонних провайдеров (не всегда)
SAML	Поддержка SSO, безопасность	Сложность настройки и управления
Kerberos	Высокая безопасность, поддержка SSO	Сложность настройки, требует синхронизации времени

1.2.2 Используемые протоколы

Игровой процесс в рамках данной работы подразумевает обмен сообщениями между клиентом и сервером в двустороннем порядке в режиме реального времени с минимальными (но не такими как при передаче аудио и видео) задержками. При этом клиенту постоянно необходимо обновлять любую изменившуюся информацию в течение всей партии, а серверу отправлять одинаковые (в большинстве случаев) сообщения всем игрокам. Исходя из вышеуказанных требований, для реализации части протокола, поддерживающей непосредственно игровой процесс оптимально использование протокола

WebSocket [5].

Преимущества протокола *WebSocket*:

— поддержка двусторонней («дуплексной») связи между клиентом и сервером;

— возможность мгновенной отправки данных клиенту (группе клиентов) без необходимости ожидания запроса;

— сокращение времени на установку соединения для отправки запросов (соединение устанавливается единожды посредством трехэтапного «рукопожатия», канал открыт для передачи сообщений до прекращения общения одним из участников процесса).

Также стоит отметить, что некоторые запросы, не связанные непосредственно с игровым процессом (авторизация/регистрация пользователей, просмотр статистики, поиск игровой комнаты) логически выходят за рамки игрового процесса и менее требовательны (общение в формате «запрос-ответ», выше лояльность к задержкам передачи данных, небольшой объем передаваемых данных в рамках запроса, отсутствие большого потока запросов от одного клиента). Но необходима гарантия доставки пакетов (целостность данных, повторная отправка). Для данных целей можно обойтись протоколом *HTTP/1.1* [6].

Также для аутентификации/авторизации пользователей в системе поверх протокола HTTP предлагается использовать имеющийся стандартный протокол *OpenID Connect* [7].

Протокол OpenID Connect

OpenID Connect (OIDC) – протокол проверки подлинности удостоверений, который является расширением открытой проверки подлинности (OAuth) 2.0 [8] и предназначен для стандартизации процесса проверки подлинности и авторизации пользователей при входе в систему для доступа к цифровым службам. OIDC обеспечивает проверку подлинности, что означает подтверждение

того, что пользователи являются теми, кем они себя называют.

OAuth 2.0 определяет, к каким системам разрешен доступ этим пользователям. OAuth 2.0 обычно используется для того, чтобы два несвязанных приложения могли обмениваться сведениями без ущерба для пользовательских данных. OAuth 2.0 основан на использовании базовых веб-технологий: HTTP-запросов, редиректов и т. п. Это делает его универсальным стандартом, который можно использовать на любой платформе с доступом к интернету и браузеру: на сайтах, в мобильных и десктопных приложениях, а также в браузерных плагинах.

В OIDC есть шесть основных компонентов:

— проверка подлинности —- процесс подтверждения того, что пользователь является тем, кем он себя называет;

— клиент — программное обеспечение, такое как веб-сайт или приложение, которое запрашивает токены, используемые для проверки подлинности пользователя или доступа к ресурсу;

— доверяющие стороны — это приложения, которые используют поставщиков OpenID для проверки подлинности пользователей;

— токены удостоверений содержат идентификационные данные, включая результат процесса проверка подлинности под, идентификатор пользователя и сведения о том, как и когда пользователь проходит проверку подлинности;

— поставщики OpenID — это приложения, для которых у пользователя уже есть учетная запись. Их роль в OIDC заключается в проверке подлинности пользователя и передаче этих сведений проверяющей стороне;

— пользователи — это люди или службы, которые пытаются получить доступ к приложению без создания новой учетной записи или предоставления имени пользователя и пароля.

Типичный процесс проверки подлинности OIDC включает следующие действия:

1. Пользователь переходит к приложению, к которому ему нужно получить доступ (проверяющая сторона).
2. Пользователь вводит свое имя пользователя и пароль.
3. Проверяющая сторона отправляет запрос поставщику OpenID.
4. Поставщик OpenID проверяет учетные данные пользователя и получает авторизацию.
5. Поставщик OpenID отправляет токен удостоверения и часто токен доступа проверяющей стороне.
6. Проверяющая сторона отправляет токен доступа на устройство пользователя.
7. Пользователю предоставляется доступ на основе сведений, предоставленных в токене доступа и проверяющей стороне.

1.2.3 Протоколы в модели OSI

Разрабатываемый протокол находится на прикладном уровне в модели OSI. Также, на рисунке 1.1 представлены используемые протоколы в модели OSI, а также протоколы нижележащих слоев, поверх которых они работают.

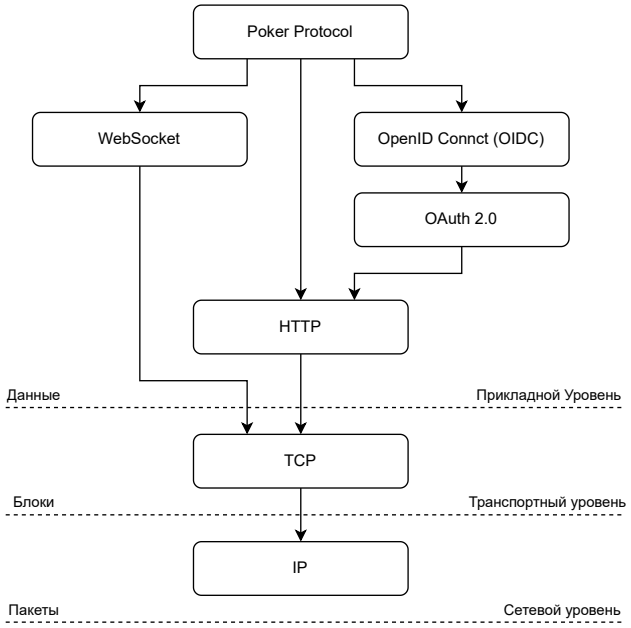

Рисунок 1.1: Используемые протоколы в модели OSI

1.3 Целевая аудитория

Целевая аудитория использования протокола, совместно с прилагаемым ПО:

— пользователи, которым не с кем поиграть в покер, но желание присутствует;

— пользователи со сформированной группой игроков, находящиеся далеко друг от друга;

— пользователи, у которых нет возможности/желания вводить роль дилера, следить за ставками и проч. (автоматизация игрового процесса);

— пользователи-участники онлайн-турниров;

— пользователи, у которых нет бумажных карт и прочих комплектующих для игры в покер.

— пользователи, предпочитающие играть в настольные игры онлайн.

Возможно использование протокола другими разработчиками для внедрения его в какие-либо проекты в роли мини-игры и проч.

2 Конструкторская часть

2.1 Участники игрового процесса и их функции

В рамках реализации задачи по разработке протокола для игры в покер задействованы некоторые ключевые элементы, выполняющие свои специфичные функции для корректной передачи и дальнейшей обработки информации в течение игрового процесса в рамках партии, а также за его пределами. Участники игрового процесса и их функции представлены ниже, на рисунке 2.1.

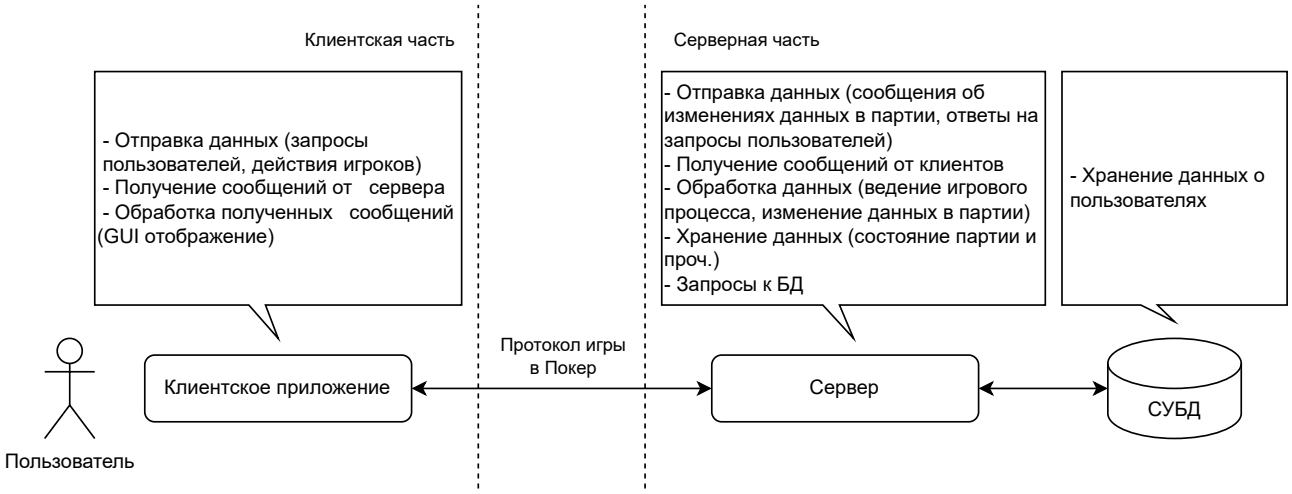

Рисунок 2.1: Участники игрового процесса и их функции

2.2 Постановка задачи

2.2.1 Уточнение правил и игрового процесса

Правила игры, которые будут использоваться на этапе разработки протокола соответствуют классическим правилам Техасского Холдема с некоторыми уточнениями. Все уточнения игрового процесса представлены далее по тексту.

Дилер, рассадка игроков, переход хода

В зависимости от локации покерного турнира, дилеры бывают двух видов: переходящие (в домашних турнирах эта роль переходит от игрока к игроку по очереди) и фиксированные (сотрудник игрового зала). Основная задача дилера – обеспечить бесперебойный игровой процесс и контролировать его ход [9].

В рамках данной работы роль дилера является комбинированной, переходит от игрока к игроку (в зависимости от рассадки) после каждого шоудауна (от положения дилера зависит с кого будут взиматься малые и большие блайнды), а все остальные действия контроля игрового процесса, такие как: тасовка и раздача карт, управление банком, распределение выигрышей и проч., – будет выполняться игровым сервером.

В начале игры дилером назначается игрок, первый севший за стол (вошедший первым в текущую игру). Рассадка игроков далее формируется следующим образом: второй вошедший в текущую игру игрок садится слева от дилера, третий – слева от второго и т.д., замыкая круг. Далее роль дилера (так же как и ход) будет передаваться по рассадке игроков. При этом в партии может участвовать от 2-х до 5-и игроков.

Банкролл, малый и большой блайнд, ставки

В рамках данной работы не предусмотрено наличие у игроков капитала, переходящего от игры к игре, но предусмотрена ранговая система достижений. В начале игры банкролл (игровой капитал) каждого игрока будет составлять 10 000 единиц игровой валюты.

Малый и большой блайнды – автоматические ставки, от которых невозможно отказаться (взимаются после раздачи карт на руки). Малый блайнд взимается с игрока, сидящего слева от дилера, большой – со следующего игрока. На первой раздаче сумма малого блайнда составляет 250 единиц игровой валюты, большого – 500. После каждого второго шоудауна размер малого блайнда возрастает на 250 единиц игровой валюты, большого – на 500. Если

у игрока на момент взимания блайнд превышает текущий капитал, игрок автоматически идет ва-банк.

В процессе торгов можно принимать ставки текущие ставки, а также повышать. Повысить ставку можно тремя способами: $\times 1.5$ от текущей ставки, $\times 2$ от текущей ставки или ва-банк. Если текущая ставка превышает текущий капитал игрока, остаться в игре можно только пойдя ва-банк.

2.2.2 Постановка задачи разработки протокола

Разработать протокол прикладного уровня онлайн-игры в покер на основе правил Техасского Холдема с учетом раздела 2.2.1 «Уточнение правил и игрового процесса» поверх протоколов WebSocket и HTTP.

Предусмотреть возможность авторизации, регистрации, доступа к статистике и данным о текущем игроке. При этом неавторизованные пользователи не смогут участвовать в игре.

Ввести для игроков следующую ранговую систему достижений, сопровождающуюся рейтингом побед (Винрейт – процент выигранных партий):

— РЕКРУТ – до 5 побед;

— РЯДОВОЙ – от 5 побед;

— СЕРЖАНТ – от 10 побед;

— КАПИТАН – от 25 побед;

— МАЙОР – от 50 побед;

— ПОЛКОВНИК – от 100 побед;

— ГЕНЕРАЛ – от 250 побед.

В течение игрового процесса (в рамках партии) для передачи данных использовать протокол WebSocket, для авторизации пользователей – OpenID

Connect, в остальных случаях – протокол HTTP. Все сообщения передавать в формате JSON [10].

2.3 Описание протокола

Ниже, в таблице 2.1 представлено описание HTTP-запросов, использующихся в работе, назначение которых будет описано далее по тексту.

Таблица 2.1: Описание запросов

URL	**Описание**
POST::/poker/v1/register	Регистрация
POST::/poker/v1/oauth/token	Авторизация
DELETE::/poker/v1/oauth/revoke	Завершение сессии
GET::/poker/v1/me	Информация о пользователе
GET::/poker/v1/rooms/matching	Поиск игровой комнаты
GET::/poker/v1/rooms/{roomUid}	Информация о комнате
GET::/poker/v1/players/{userUid}	Информация об игроке
GET::/poker/v1/rooms-ws/{roomUid}	Переход на WebSocket

2.3.1 Регистрация и авторизация пользователей

Чтобы принять участие в игре, пользователю необходимо пройти процедуру авторизации (при наличии аккаунта) или же регистрации (создать новый аккаунт). Данные процедуры проводятся через HTTP запросы, где на стороне сервера используется протокол OIDC. Схемы авторизации и регистрации представлены ниже, на рисунках 2.2 и 2.3, с подробной информацией о запросах в таблицах 2.2 - 2.7.

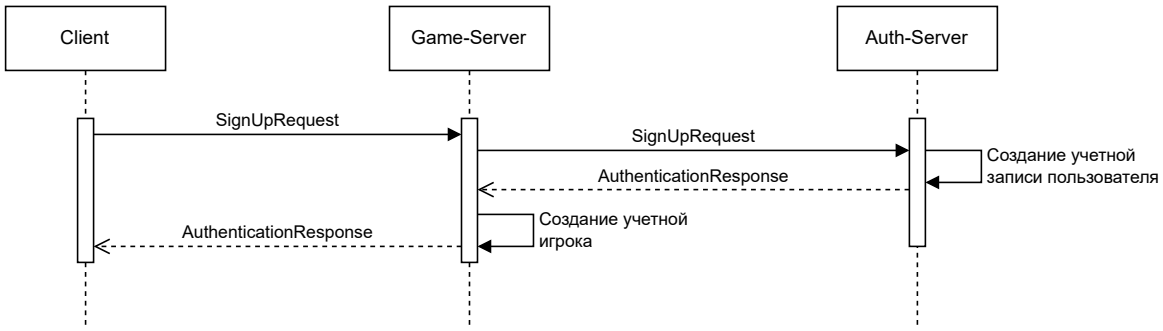

Рисунок 2.2: Регистрация (POST */poker/v1/register*)

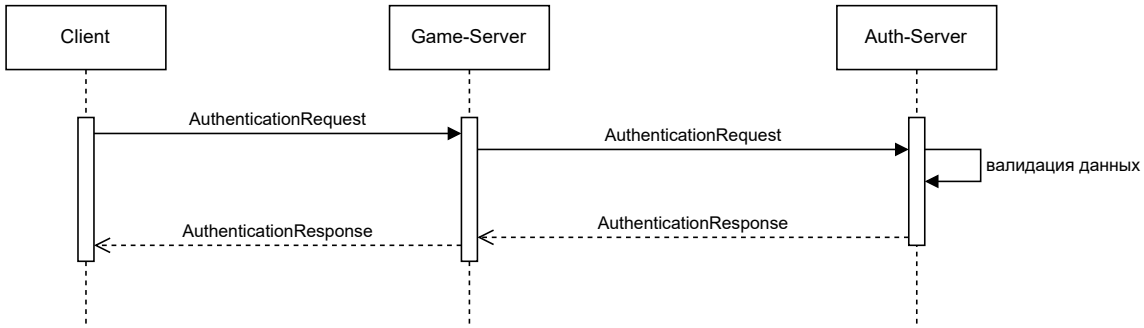

Рисунок 2.3: Авторизация (POST */poker/v1/oauth/token*)

ПРИМЕЧАНИЕ: далее по тексту в таблицах, содержащих в себе структуру сообщения (запроса/ответа) в графе «Поле» звездочкой указаны необходимые поля сообщений, а в Описании для необязательных полей после символа звездочка следует название поля и его значение, при котором рассматриваемое поле передается в теле сообщения.

Таблица 2.2: Тело *SignUpRequest*

Поле	**Тип**	**Описание**
Scope*	str	'OPENID'
Username*	str	Имя пользователя
Password*	str	Пароль

Таблица 2.3: Ответ на запрос регистрации

Статус код	**Ответ**	**Описание**
200	AuthenticationResponse	Сессионные параметры
400	ErrorResponse	Пользователь с таким логином уже существует
500	ErrorResponse	Внутренняя ошибка сервера

Таблица 2.4: Тело *AuthenticationResponse*

Поле	**Тип**	**Описание**
UserUid*	UUID	Идентификатор пользователя
RefreshToken*	JWT	Рефреш-токен
AccessToken*	JWT	Акксесс-токен
ExpiresIn*	int	Время прекращения действия акксесс-токена
Scope*	str	'OPENID'

Scope в контексте OIDC – определяющий запрашиваемый доступ ресурс или информация о пользователе, к которым можно получить доступ.

Общие области включают «openid» (обязательное), «profile» и «email» и т.п. При запросе авторизации клиент может указать необходимый scope в соответствии с потребностями приложения. В данной работе *Scope* – 'OPENID' ставит в обязательную область поля, указанные в таблице 2.4.

Акксесс-токен – подтверждение того, что пользователь является тем, за кого он себя выдает. Имеет определенный срок действия, в течение которого пользователь может отправлять сообщения серверу с гарантией его подлинности при предъявлении акксесс-токена. После истечения его срока действия для дальнейшей возможности отправки сообщений необходимо пройти аутентификацию, предоставив рефреш-токен, по которому произойдет выдача нового акксесс-токена и обновление рефреш-токена (таблица 2.5).

У рефреш-токена также есть свой срок действия (как правило, многократно больше акксесс-токена), после которого снова авторизоваться можно будет только с помощью ввода логина и пароля.

Таблица 2.5: Тело *AuthenticationRequest*

Поле	**Тип**	**Описание**
Scope*	str	'OPENID'
GrantType*	str	'PASSWORD'/'REFRESH-TOKEN' (тип авторизации)
Username	str	Имя пользователя **GrantType* – 'PASSWORD'
Password	str	Пароль **GrantType* – 'PASSWORD'
RefreshToken	JWT	Рефреш-токен **GrantType* – 'REFRESH-TOKEN'

Таблица 2.6: Ответ на запрос авторизации

Статус код	**Ответ**	**Описание**
200	AuthenticationResponse	Сессионные параметры
401	ErrorResponse	Неверное имя пользователя/ пароль, истек акксесс-токен
500	ErrorResponse	Внутренняя ошибка сервера

Таблица 2.7: Тело *ErrorResponse*

Поле	**Тип**	**Описание**
Message*	str	Описание ошибки

ПРИМЕЧАНИЕ: далее по тексту структура *ErrorResponse* идентична.

В данной работе формат акксесс- и рефреш-токенов определен стандартом JSON Web Token (JWT). Структура токенов представлена ниже, в таблице 2.8.

Таблица 2.8: Структура *JWT-токена*

Header	**Тип**	**Описание**
Alg	str	Алгоритм подписи
Typ	str	'JWT'
Kid	str	Идентификатор ключа, которым подписали токен
TokenType	str	'ACCESS'/'REFRESH'
TokenPayload	**Тип**	**Описание**
Jti	str	Идентификатор токена
UserUid	UUID	Идентификатор пользователя
Iss	str	Автор токена
Iat	int64	Время выпуска токена
Exp	int64	Время окончания действия токена
DeviceID	UUID	Идентификатор устройства

После успешной авторизации/регистрации пользователь может получить данные своего аккаунта посредством запроса информации о себе (рисунок 2.4), подробнее в таблицах 2.9 и 2.10.

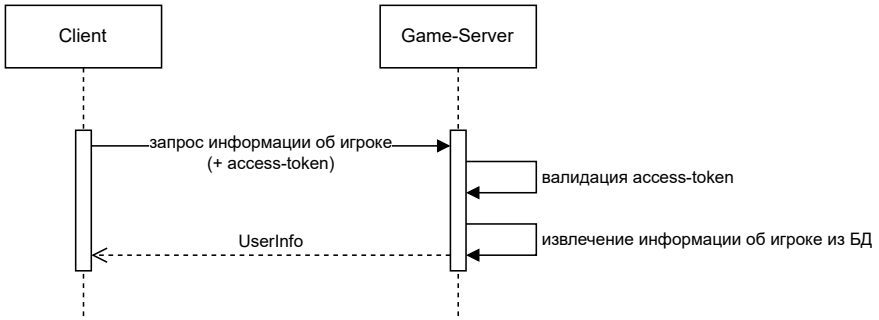

Рисунок 2.4: Получение информации об игроке (GET */poker/v1/me* headers={'Authorization': AccessToken})

Таблица 2.9: Ответ на запрос о получении информации о пользователе

Статус код	**Ответ**	**Описание**
200	UserInfo	Информация об игроке
401	ErrorResponse	Истек акксесс-токен
500	ErrorResponse	Внутренняя ошибка сервера

Таблица 2.10: Тело *UserInfo*

Поле	**Тип**	**Описание**
UserUid*	UUID	Идентификатор пользователя
Username*	str	Имя пользователя
NumOfGames*	int	Количество сыгранных игр
NumOfWins*	int	Количество побед
UserRank*	str	'РЕКРУТ'/'РЯДОВОЙ'/'СЕРЖАНТ' /'КАПИТАН'/'МАЙОР'/ 'ПОЛКОВНИК'/'ГЕНЕРАЛ'
UserState*	str	'MENU'/'IN-GAME'
RoomUid	UUID	Идентификатор комнаты **UserState* – 'IN-GAME'

2.3.2 Поиск игровой комнаты и подключение

После успешной авторизации/регистрации пользователь может инициировать процесс игры, нажав на кнопку «Играть» в приложении. Далее запускается процесс поиска комнаты посредством HTTP-запроса, представленного на рисунке 2.5.

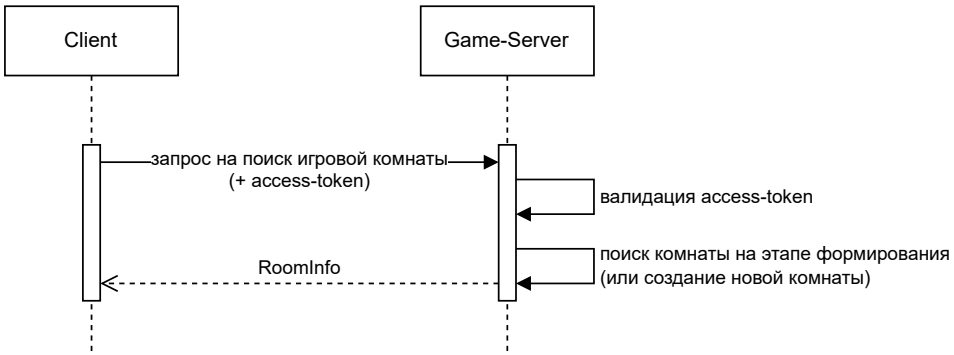

Рисунок 2.5: Поиск игровой комнаты (GET */poker/v1/rooms/matching* headers={'Authorization': AccessToken})

При этом, если имеется комната, статус которой – 'FORMING' (число игроков не больше 5 и суммарный ответ на опрос игроков в комнате до начала игрового процесса о начале игры – 'ОЖИДАНИЕ'), то игрок направляется в данную комнату, иначе – формируется новая комната и игрок направляется в нее. Жизненный цикл комнаты представлен на рисунке 2.6.

Рисунок 2.6: Жизненный цикл игровой комнаты

В качестве результата поиска игровой комнаты игровой сервер присылает клиенту информацию о состоянии комнаты в виде JSON-объекта *RoomInfo*. Под состоянием комнаты понимается данные о состоянии игры и ее участников внутри комнаты от лица конкретного игрока. Структура объекта *RoomInfo* описана в таблицах 2.11–2.15.

Таблица 2.11: Ответ на запрос о поиске игровой комнаты

Статус код	**Ответ**	**Описание**
200	RoomInfo	Состояние игры в комнате, в которую распределен пользователь
401	ErrorResponse	Истек акксесс-токен
500	ErrorResponse	Внутренняя ошибка сервера

Таблица 2.12: Тело *RoomInfo*

Поле	**Тип**	**Описание**
RoomUid*	UUID	Идентификатор комнаты
RoomState*	str	'FORMING'/'GAMING'/ 'DISSOLUTION' Статус комнаты
NumOfPlayers*	int	Количество игроков в комнате
NumOfStartPlayers*	int	Количество игроков, проголосовавших за начало игры
PlayerList*	list[PlayerInfo]	Список игроков в комнате
TableCardList	list[PlayingCard]	Список открытых карт на столе **RoomState* – 'GAMING'
Stack	int	Сумма ставок пользователей (банк раунда) **RoomState* – 'GAMING'
Dealer	UUID	UserUid игрока, у которого на данный момент роль дилера **RoomState* – 'GAMING'
Bout	UUID	UserUid игрока, которому на данный момент принадлежит ход **RoomState* – 'GAMING'
LastEventId	int	Идентификатор последнего события **RoomState* – 'GAMING'
RoundNumber	int	Номер раунда **RoomState* – 'GAMING'

Таблица 2.13: Тело *PlayerInfo*

Поле	**Тип**	**Описание**
UserUid*	UUID	Идентификатор пользователя
Username*	str	Имя пользователя
Bet*	int	Текущая ставка игрока
Deposit*	int	Текущий банк игрока
LastActionLabel*	str	'NONE'/'FOLD'/'CHECK' /'CALL'/'RAISE'/'ALL-IN' Последнее действие игрока
UserRank*	str	'РЕКРУТ'/'РЯДОВОЙ'/'СЕРЖАНТ' /'КАПИТАН'/'МАЙОР'/ 'ПОЛКОВНИК'/'ГЕНЕРАЛ' Ранг игрока
PersonalCardList*	list[PlayingCard]	Список карт игрока
BestCombName*	str	'РОЯЛ-ФЛЕШ'/'СТРИТ-ФЛЕШ'/'КАРЕ'/'ФУЛЛ-ХАУС'/'ФЛЕШ'/'СТРИТ'/ 'ТРОЙКА'/'ДВЕ ПАРЫ'/'ПАРА'/'ВЫСШАЯ КАРТА' Название наилучшей возможной комбинации у игрока
BoutVariants*	list[BoutVariant]	'FOLD'/'CHECK' /'CALL'/'RAISE' Возможные варианты хода игрока, когда ему принадлежит ход
TimeEndBoutOrForming*	int	Момент времени (в милисекундах) завершения хода игрока или завершения стадии формирования комнаты
VoteType*	str	'WAIT'/'START' Голос игрока: дождаться присоединения других игроков или начать игру немедленно (По умолчанию – 'WAIT', можно изменить в любой момент времени)

Таблица 2.14: Тело *PlayingCard*

Поле	**Тип**	**Описание**
CardSuit*	str	'DIAMONDS'/'HEARTS'/'CLUBS'/'SPADES' Масть карты
Index*	str	'2'/'3'/'4'/'5'/'6'/'7'/'8'/'9'/'10'/'JACK'/ 'QUEEN'/'KING'/'ACE' Номинал карты

Таблица 2.15: Тело *BoutVariant*

Поле	**Тип**	**Описание**
VariantType*	str	'FOLD'/'CHECK'/'CALL'/'RAISE' Вариант хода
CallValue	int	Значение принимаемой ставки **VariantType* – 'CALL'
RaiseVariants	str	'X1_5'/'X2'/'ALL-IN' Вариант увеличения ставки **VariantType* – 'RAISE'

После получения *RoomUid* (из объекта RoomInfo) с клиента автоматически отправляется HTTP-запрос GET */poker/v1/rooms-ws/{roomUid}* headers = {'Upgrade': 'WebSocket'} для перехода на протокол WebSocket и установления соединения. Описание ответов на запрос приведено в таблице 2.16.

Таблица 2.16: Ответ на запрос о подключении к игровой комнате

Статус код	**Ответ**	**Описание**
101		Переход на протокол WebSocket
404	ErrorResponse	Комната не найдена
500	ErrorResponse	Внутренняя ошибка сервера

Для проверки активности WebSocket-соединения используется передача ping-, pong-сообщений, структура которых описана в таблице 2.17. Игровой сервер отправляет клиенту ping-сообщение каждые 500 мс. Клиент отвечает на ping pong-сообщением. Если сервер на протяжении 500мс (+100мс на задержку) не получает от клиента ни одного сообщения любого типа (остальные типы сообщений описаны далее), то он закрывает WebSocket-соединие. Аналогично поступает клиент. Далее клиент действует в соответствии со сценарием потери соединения (см. раздел 2.3.4).

Таблица 2.17: Тело *Ping- и PongMessage*

Поле	**Тип**	**Описание**
MessageType*	str	'PING'/'PONG'

Далее со стороны клиента происходит аутентификация соединения WebSocket путем отправки по нему сообщения в формате JSON *AuthMessage*, содержащего access-token игрока. Описание структуры AuthMessage приведено

в таблице 2.18. Значение поля *LastEventId* берется из одноименного поля объекта *RoomInfo*, полученного в качестве ответа на запрос о поиске игровой комнаты или при запросе состояния игровой комнаты (см. раздел 2.3.4 «Обработка потери соединения»). Игровой сервер валидирует полученный токен. При успешной проверке сервер передает по открытому WebSocket-соединению ответ в виде JSON-объекта *ResponseMessage* со значением *StatusCode* 200. Описание структуры AuthMessage приведено в таблице 2.19.

Игровой сервер сохраняет значение времени истечения срока действия access-token. Затем последовательно одно за другим игровой сервер пересылает клиенту сообщения о событиях, которые появились после события идентификатор которого был передан в поле *LastEventId*. Далее происходит игровое взаимодействие клиента и игрового сервера, которое будет описано далее в разделе 2.3.3 «Игровое взаимодействие».

Когда истекает срок действия access-token, для продолжения взаимодействия с игровым сервером клиент обновляет access-token с помощью отправки запроса *AuthenticationRequest* на аутентификацию по refresh-token. При успешной повторной аутентификации клиент получает новую пару токенов. Далее процедура аутентификации WebSocket-соединения повторяется. До тех пор, пока клиент не аутентифицирует WebSocket-соединение игровой сервер не будет отправлять ему сообщения, а полученные от клиента сообщения не будут обработаны сервером. Описание последовательности действий при поиске комнаты и подключении к ней приведено на рисунке 2.7. Взаимодействия в рамках игры между игровым сервером и клиентами будут происходить по открытым WebSocket-соединениям.

Таблица 2.18: Тело *AuthMessage*

Поле	**Тип**	**Описание**
MessageType*	str	допустимое значение: 'AUTH' Тип сообщения
MessageId*	int	Идентификатор сообщения
RoomUid*	UUID	Идентификатор комнаты, к которой присоединился игрок
Token*	str	Акксесс-токен клиента
LastEventId*	int	Идентификатор последнего события игровой комнаты

Таблица 2.19: Тело *ResponseMessage*

Поле	**Тип**	**Описание**
MessageType*	str	допустимое значение: 'ACK' Тип сообщения
AckMessageId*	int	Идентификатор сообщения, для которого является ответом данное сообщение
StatusCode*	int	Код результата. Возможные значения: 200 - сообщение обработано корректно; 401 - срок действия access-token истек; 400 - некорректная команда

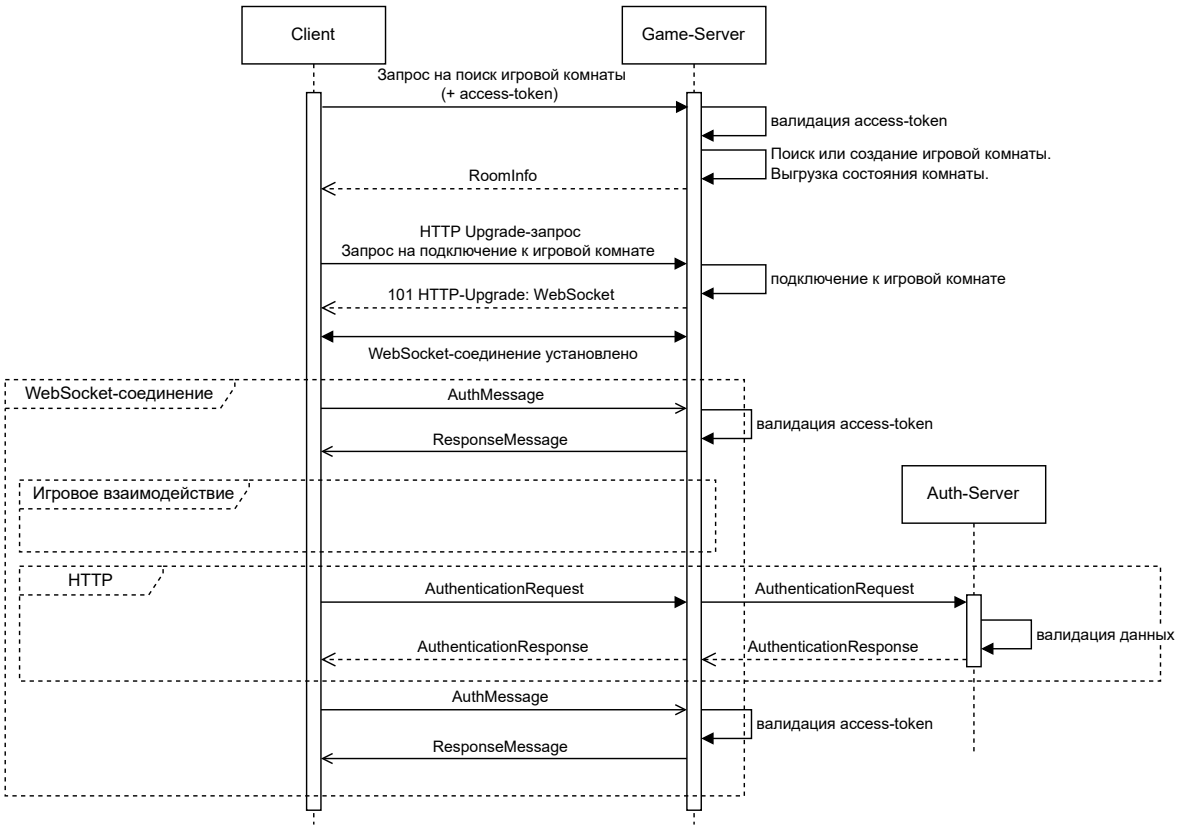

Рисунок 2.7: Поиск игровой комнаты и подключение к ней

2.3.3 Игровое взаимодействие

Игровое взаимодействие строится на том, что игровой сервер ведет обсчет партии и состояний игроков, в ней участвующих, и сообщает участникам партии об изменениях в ней путем рассылки сообщений *EventMessage* клиентам по WebSocket-соединениям. Описание структуры *EventMessage* приведено в таблице 2.20. Следует обратить внимание на поле *EventDescriptor* в этой структуре. Оно содержит описание события, которое может быть одним из трех типов:

1. *PrepareEvent*: содержит информацию об обновленных результатах голосования перед началом игры (см. далее раздел «Стадия формирования»).

2. *GameEvent*: содержит информацию о событиях комнаты: изменение статуса комнаты, распределение закрытых карт по игрокам, выкладывание открытых карт и т.д. (подробнее см. таблицу 2.22).

3. *PlayerActionEvent*: содержит информацию о действиях игроков комнаты: присоединение или покидание игроком комнаты, тип хода игрока и т.д. (подробнее см. таблицу 2.23).

Полные описания структур дескрипторов событий приведены в таблицах 2.20–2.23.

Таблица 2.20: Тело *EventMessage*

Поле	**Тип**	**Описание**
MessageType*	str	Допустимое значение: 'EVENT'; тип сообщения
MessageId*	int	Идентификатор сообщения
EventType*	str	'PREPARE-EVENT' / 'GAME-EVENT' / 'PLAYER-ACTION-EVENT'
EventDescriptor*	Object	*PrepareEvent* / *GameEvent* *PlayerActionEvent* Дескриптор события

Таблица 2.21: Тело *PrepareEvent*

Поле	**Тип**	**Описание**
EventId*	int	Идентификатор события
NumOfPlayers*	int	Количество игроков в комнате
NumOfStartPlayers*	int	Количество игроков, проголосовавших за начало игры

Таблица 2.22: Тело *GameEvent*.

Поле	**Тип**	**Описание**
EventId*	int	Идентификатор события
EventType*	str	'ROOM_STATE_UPDATE' – обновление статуса комнаты 'PERSONAL_CARDS' – сообщение содержит список карт игрока, выданных взакрытую в начале раунда 'CARDS_ON_TABLE' – обновление списка открытых карт, лежащих на столе 'NEW_ROUND' – начало нового раунда (Метки последних действий всех игроков становятся NONE) 'NEW_TRADE_ROUND' – начало нового круга торгов внутри раунда (метки последних действий всех игроков (кроме тех, кто выбрал 'FOLD' или 'ALL-IN') становятся NONE) 'BET_ACCEPTED' – ставки приняты, фишки игроков переносятся в банк 'WINNER_RESULT' – вскрытие всех карт игроков, оглашение выигравшей комбинации и победителя(-ей) раунда

Продолжение таблицы 2.22		
Поле	**Тип**	**Описание**
NewRoomState	str	'FORMING'/'GAMING'/'DISSOLUTION' Новый статус комнаты **EventType* – 'ROOM_STATE_UPDATE'
RoundNumber	int	Номер раунда **EventType* – 'NEW_ROUND'
PlayingCardsList	list[PlayingCard]	Список карт игрока или список открытых карт на столе **EventType* – 'PERSONAL_CARDS'/ 'CARDS_ON_TABLE')
ClosedCards	list[list[PlayingCard]]	В i-й ячейке содержит карты i-го игрока поля PlayerUids **EventType* – 'WINNER_RESULT'
PlayerUids	list[UUID]	Идентификаторы игроков комнаты **EventType* – 'WINNER_RESULT'
WinnerUids	list[UUID]	Идентификаторы победителей(-я) раунда **EventType* – 'WINNER_RESULT'
BestCombinations	list[PlayingCard]	В i-й ячейке содержатся карты выигрышной комбинации i-го победителя из WinnerUids **EventType* – 'WINNER_RESULT'

Продолжение таблицы 2.22		
Поле	**Тип**	**Описание**
BestCombName	str	’РОЯЛ-ФЛЕШ’/’СТРИТ-ФЛЕШ’/’КАРЕ’/’ФУЛЛ-ХАУС’/’ФЛЕШ’/’СТРИТ’/ ’ТРОЙКА’/’ДВЕ ПАРЫ’/’ПАРА’/’ВЫСШАЯ КАРТА’ Название выигрышной комбинации раунда **EventType* – ’WINNER_RESULT’
WinnerDeposits	list[int]	В i-й ячейке содержит обновленный депозит i-го победителя из WinnerUids **EventType* – ’WINNER_RESULT’
NewStack	int	Обновленное (обнуленное) значение банка на столе **EventType* – ’WINNER_RESULT’

Таблица 2.23: Тело *PlayerActionEvent*.

Поле	**Тип**	**Описание**
EventId*	int	Идентификатор события
UserUid*	UUID	Идентификатор игрока, совершившего действие

Продолжение таблицы 2.23		
Поле	**Тип**	**Описание**
ActionType*	str	'INCOME' - игрок присоединился к комнате 'OUTCOME' - покинул комнату 'BOUT' - получил черед хода 'FOLD'/'CHECK'/'CALL'/ 'RAISE'/'ALL-IN' - сделал соответствующее действие в процессе игры 'SET-DEALER' - игрок получает статус дилера 'MIN-BLIND-IN' - игрок вносит малый блайнд 'MAX-BLIND-IN' - игрок вносит большой блайнд (Если *ActionType* – 'OUTCOME' и в *UserUid* находится UUID текущего игрока, значит он проиграл)
BoutVariants	list[BoutVariant]	Возможные варианты хода игрока **ActionType* – 'BOUT'
BestCombName	str	'РОЯЛ-ФЛЕШ'/'СТРИТ-ФЛЕШ'/'КАРЕ'/'ФУЛЛ-ХАУС'/'ФЛЕШ'/'СТРИТ'/ 'ТРОЙКА'/'ДВЕ ПАРЫ'/'ПАРА'/'ВЫСШАЯ КАРТА' Название лучшей комбинации игрока **ActionType* – 'BOUT'

Продолжение таблицы 2.23		
Поле	**Тип**	**Описание**
TimeEndBoutOrForming	int	Момент времени (в миллисекундах) завершения очереди хода игрока или завершения стадии формирования комнаты *$ActionType$ – 'INCOME' или 'OUTCOME' или 'BOUT'
NewBet	int	Обновленное значение ставки игрока *$ActionType$ – 'CALL'/'RAISE'
NewDeposit	int	Обновленное значение депозита игрока *$ActionType$ – 'CALL'/'RAISE'

Клиенты сообщают игровому серверу о действиях игроков путем отправки сообщений *ActionMessage* по WebSocket-соединению. Структура таких сообщений описана в таблице 2.24. Игровой сервер принимает сообщение, проверяет аутентификацию соединения, корректность сообщения и возвращает обратно клиенту результат в *ResponseMessage* (см. таблицу 2.19). Примеры корректных сообщений, присылаемых клиентами для каждой стадии жизненного цикла комнаты, приведены ниже в описании каждой стадии.

Далее игровой сервер по WebSocket-соединениям рассылает всем клиентам, подключенным к комнате, информацию о действии игрока.

Таблица 2.24: Тело *ActionMessage*

Поле	**Тип**	**Описание**
MessageType*	str	'GAME-ACTION' / 'VOTE' тип сообщения
MessageId*	int	Идентификатор сообщения
RoomUid*	UUID	Идентификатор игровой комнаты, к которой присоединен игрок
UserUid*	UUID	Идентификатор игрока, совершающего действие
ActionType	str	'FOLD'/'CHECK'/'CALL'/'RAISE'/'OUTCOME' Тип действия игрока **MessageType* 'GAME-ACTION'
Coef	str	'X1_5'/'X2'/'ALL-IN' варианты повышения ставки **ActionType* 'RAISE'
VoteType	str	'START'/'WAIT' Тип голоса игрока **MessageType* 'VOTE'

Как было описано выше (см. рисунок 2.6), игровая комната может иметь один из трех статусов: FORMING (стадия формирования), GAMING (стадия игрового процесса), DISSOLUTION (расформирование комнаты).

Стадия формирования

На стадии формирования игроки присоединяются к комнате. Когда в комнате набирается два и более игроков, включается таймер ожидания на 90с. Если после запуска таймера в комнате снова стало менее двух игроков, таймер останавливается и сбрасывается (когда снова два и более запускается заново).

Уже присоединившиеся ожидают пока не выполнятся условия перехода в стадию игрового процесса: в комнате должно набраться пять человек, или больше половины всех игроков (но не менее двух) проголосуют за начало игры или истекут 90с на запущенном таймере.

При подключении игрока сервер рассылает всем игрокам *EventMessage* (с *PlayerActionEvent* в дескрипторе), пример которого приведен в листинге 2.1. В поле *timeEndBoutOrForming* приходит значение момента времени (в милисекундах) в формате UNIX-TIME, когда начнется игра, если таймер запущен. Иначе данное поле будет хранить ноль.

```
{
  "MessageType": "EVENT",
  "MessageId": 1739459149110833100,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
    "EventId": 1739461989355181600,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "INCOME",
        "TimeEndBoutOrForming": 1739461989355
  }
}
```

Листинг 2.1: Пример *EventMessage* при присоединении игрока

При получении от сервера сообщения о присоединении нового игрока клиент делает HTTP-запрос (*GET /poker/v1/player2/{userUid} headers={ 'Authorization' : AccessToken}*). В качестве ответа клиент получает объект PlayerInfo с информацией о пользователе (см. таблицу 2.13; коды при обработке запроса аналогичны описанным в таблице 2.9).

Каждый игрок может проголосовать за «ожидание» (то есть ждать, пока наберется в комнате 5 игроков) или за «начало» (начать игру, не дожидаясь полного заполнения комнаты) и менять свой голос неограниченное количество раз, пока не завершится стадия формирования. По умолчанию, считается, что каждый игрок проголосовал за «ожидание». В листинге 2.3 приведен пример *ActionMessage*, присылаемого клиентом для передачи голоса игрока. В результате обработки такого сообщения игровой сервер отправит всем клиентам, **EventMessage** (с *PrepareEvent* в дескрипторе), приведенный в листинге 2.3 (если предположить, что к комнате было присоединено три игрока и ни один из них ранее не проголосовал за «начало»).

```
{
  "MessageType": "VOTE",
  "MessageId": 1739458972617012800,
  "RoomUid": "4149a6ca-1052-4aed-8202-b43a804a56d6",
  "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
  "VoteType": "START"
}
```

Листинг 2.2: Пример *ActionMessage* для передачи голоса игрока

```
{
  "MessageType": "EVENT",
  "MessageId": 1534639288113431005,
    "EventType": "PREPARE-EVENT",
  "EventDescriptor": {
    "EventId": 1739463981680720000,
```

```
    "NumOfPlayers": 3,
    "numOfStartPlayers": 1
  }
}
```

Листинг 2.3: Пример *EventMessage* об изменении соотношения голосов

Также игрок в любой момент может покинуть комнату, в этом случае от клиента придет *ActionMessage*, пример которого приведен в листинге 2.4, далее клиент закроет WebSocket-соединение; а игровой сервер пришлет остальным клиентам, присоединенной к комнате *EventMessage* (с *PlayerActionEvent* в дескрипторе), пример которого приведен в листинге 2.5. Затем игровой сервер отправит сообщение, аналогичное тому, что приведено в листинге 2.3.

```
{
  "MessageType": "GAME-ACTION",
  "MessageId": 1739464227684035900,
  "RoomUid": "4149a6ca-1052-4aed-8202-b43a804a56d6",
  "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
  "ActionType": "OUTCOME"
}
```

Листинг 2.4: Пример *ActionMessage* о выходе игрока из комнаты

```
{
  "MessageType": "EVENT",
  "MessageId": 1739459149110833100,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
    "EventId": 1739461989355181600,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "OUTCOME",
        "TimeEndBoutOrForming": 0
  }
}
```

Листинг 2.5: Пример *EventMessage* о выходе игрока из комнаты

Остальные виды сообщений, присылаемые клиентами на этой стадии, являются некорректными и не будут влиять на состояние комнаты.

Когда условия перехода в стадию игрового процесса будут выполнены, игровой сервер присылает всем клиентам, подключенным к комнате сообщения *EventMessage* (с *GameEvent* в дескрипторе) об изменении статуса комнаты. Пример сообщения приведен в листинге 2.6.

```
{
  "MessageType": "EVENT",
```

```
  "MessageId": 9459149110833100,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
         "EventId": 1984,
         "EventType": "ROOM_STATE_UPDATE",
          "NewRoomState": "GAMING"
  }
}
```

Листинг 2.6: Пример *EventMessage* об изменении состояния комнаты

Игровой процесс

В начале игры у каждого из игроков одинаковое количество игровой валюты. Игровой процесс делится на раунды. Каждый раунд состоит из четырех кругов торгов, в рамках которых игроки делают ставки, формируя банк раунда. В конце раунда банк распределяется между победителями поровну. Игрок, оказавшийся с нулевым депозитом на момент завершения раунда, покидает комнату. Игра длится до тех пор, пока в комнате не останется один игрок.

В начале каждого раунда игровой сервер присылает всем клиентам *EventMessage* (с *GameEvent* в дескрипторе) с номером раунда. Пример сообщения приведен в листинге 2.7.

```
{
  "MessageType": "EVENT",
  "MessageId": 9459149110833100,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
          "EventId": 482,
         "EventType": "NEW_ROUND",
         "RoundNumber": 1
  }
}
```

Листинг 2.7: Пример *EventMessage* о начале нового раунда

В начале каждого раунда назначается дилер. В первом раунде им становится игрок, первый присоединившийся к комнате (из оставшихся). Далее каждый новый раунд дилером становится следующий игрок по очередности присоединения. После последнего игрока в данной очередности круг замыкается и дилером назначается первый игрок. При назначении дилера игровой сервер присылает игрокам *EventMessage* (с *PlayerActionEvent* в дескрипторе), пример которого приведен в листинге 2.8.

```
{
  "MessageType": "EVENT",
  "MessageId": 1739459149110833100,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
    "EventId": 1739461989355181600,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "SET-DEALER"
  }
}
```

Листинг 2.8: Пример *EventMessage* о назначении игрока дилером

В начале каждого круга торгов игровой сервер присылает всем клиентам *EventMessage* (с *GameEvent* в дескрипторе), уведомляя о начале нового круга торгов. Пример сообщения приведен в листинге 2.9.

```
{
  "MessageType": "EVENT",
  "MessageId": 9459149110833100,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
         "EventId": 483,
         "EventType": "NEW_TRADE_ROUND",
  }
}
```

Листинг 2.9: Пример *EventMessage* о начале нового круга торгов

На первом круге торгов каждого раунда игрок, следующий по очередности присоединения за дилером вносит малый блайнд, а следующий после него вносит большой блайнд (как и в случае переназначений дилера данная очередность рассматривается как цикличная).

При внесении блайндов игровой сервер рассылает клиентам, подключенным к комнате, *EventMessage* (с *PlayerActionEvent* в дескрипторе). Пример сообщения о внесении малого блайнда приведен в листинге 2.10. Размер малого блайнда в начале игры составляет 250 единиц игровой валюты, далее каждые два раунда, увеличивается на 250 единиц. Большой блайнд равен двум малым.

Если размер депозита игрока меньше, размера малого или большого блайнда, то игрок автоматически совершает действие ALL-IN: игровой сервер рассылает всем клиентам, подключенным к комнате, сообщение, аналогичное приведенному в листинге 2.14

```
{
  "MessageType": "EVENT",
  "MessageId": 1,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
    "EventId": 2,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "MIN-BLIND-IN"
        "NewBet": 250
  }
}
```

Листинг 2.10: Пример *EventMessage* о внесении игроком малого блайнда

Далее на первом круге торгов каждого раунда происходит раздача игрокам карт взакрытую: каждый получает по две карты. Игровой сервер рассылает клиентам, подключенным к комнате, *EventMessage* (с *GameEvent* в дескрипторе). Каждый игрок видит свои карты и не видит карты других игроков. Пример сообщения приведен в листинге 2.11.

```
{
  "MessageType": "EVENT",
  "MessageId": 9100,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
      "EventId": 564,
        "EventType": "PERSONAL_CARDS",
      "PlayingCardsList": [
            {
                "CardSuit": "DIAMONDS",
                "Index": "JACK"
            },
            {
                "CardSuit": "DIAMONDS",
                "Index": "QUEEN"
            }]
  }
}
```

Листинг 2.11: Пример *EventMessage* с распределением карт взакрытую

После раздачи карт назначается черед хода игроку, следующему после игрока вносившего большой блайнд в этом раунде (на втором и последующих кругах торгов первым ходит игрок, следующий после дилера). Для передачи очередности хода игровой сервер присылает клиенту игрока, которому передается черед, *EventMessage* (с *PlayerActionEvent* в дескрипторе), пример которого приведен в листинге 2.12. Перед отправкой этого сообщения сервер на основе значений о ставках и депозите игрока определяет возможные

операции для этого игрока. Эти операции описаны в поле *BoutVariants*. Также игровой сервер определяет и записывает в поле *BestCombName* наиболее сильную комбинацию, которую можно получить из конфигурации карт на столе и имеющихся у игрока.

Другим клиентам, подключенным к комнате, игровой сервер отправляет сообщение, приведенное в листинге 2.13. Как видно из листинга, у него будет то же значение поля *EventId* (так как это одно и то же событие, но с точки зрения других игроков), но нет полей *BoutVariants* и *BestCombName*.

```
{
  "MessageType": "EVENT",
  "MessageId": 9101,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
      "EventId": 3,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "BOUT",
        "BoutVariants": [
            {
                "VariantType": "FOLD"
            },
            {
                "VariantType": "CALL",
                "CallValue": 500
            },
            {
                "VariantType": "RAISE",
                "RaiseVariants": ["X1_5", "X2", "ALL-IN"]
            }],
        "BestCombName": "РОЯЛ-ФЛЕШ",
        "TimeEndBout": 1739476551318
  }
}
```

Листинг 2.12: Пример *EventMessage* о передаче череда хода игроку для самого игрока

```
{
  "MessageType": "EVENT",
  "MessageId": 9101,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
      "EventId": 3,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "BOUT",
        "TimeEndBout": 1739476551318
  }
}
```

Листинг 2.13: Пример *EventMessage* о передаче череда хода игроку для других игроков игрока

Игрок, получивший очередь хода, может сделать одно из действий, описанных в массиве поля *BoutVariants*. На ход игроку дается одна минута. В результате этого клиент игрока, имеющего черед хода в данный момент, отправляет *ActionMessage*. Действия, не входящие в этот массив, или, полученные от других клиентов (игрокам которых в момент получения сообщения игровым сервером ход не принадлежал), или полученные после истечения отведенного на ход времени, являются некорректными и на состояние комнаты влиять не будут. Если в течение отведенного на ход времени игрок не сделал ход, то сервер по умолчанию выбирает вариант 'CHECK' (либо, если он невозможен, то 'FOLD'). Пример корректного в данном случае сообщения о действии приведен в листинге 2.14.

```
{
    "MessageType": "GAME-ACTION",
  "MessageId": 112,
    "RoomUid": "4149a6ca-1052-4aed-8202-b43a804a56d6",
  "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
  "ActionType": "RAISE",
    "Coef": "ALL-IN"
}
```

Листинг 2.14: Пример *ActionMessage* с информацией о выбранном действии

Получив корректный ответ о действии игрока, игровой сервер обновляет значение ставки и депозита игрока и рассылает всем подключенным к комнате клиентам *EventMessage* (с *PlayerActionEvent* в дескрипторе), пример которого приведен в листинге 2.15. В полях *NewBet* и *NewDeposit* передаются обновленные значения ставки игрока и его депозита соответственно.

```
{
  "MessageType": "EVENT",
  "MessageId": 12,
    "EventType": "PLAYER-ACTION-EVENT",
  "EventDescriptor": {
    "EventId": 13,
    "UserUid": "a0c85419-fa1c-449d-b828-1760a1be0c1a",
    "ActionType": "ALL-IN",
        "NewBet": 10000,
        "NewDeposit": 0
  }
}
```

Листинг 2.15: Пример *EventMessage*, уведомляющий о действии игрока

Далее черед хода передается следующему (по очередности присоединения к комнате) игроку. При передаче очереди игроку, текущая ставка которого

равна наибольшей ставке текущего круга торгов, (это игрок последним на текущем круге торгов поднимал ставку) в поле *BoutVariants* вместо 'FOLD' и 'CALL' доступна операция 'CHECK'. Если игрок выбрал действие 'FOLD' или 'ALL-IN', то к нему черед хода не передается до завершения текущего раунда.

После того как игрок, которому доступна операция 'CHECK', ее совершает; игровой сервер принимает ставки: ставки игроков обнуляются, а банк игры пополняется на сумму ставок всех игроков, сделанных на данном круге торгов. Сервер рассылает всем подключенным к комнате клиентам *EventMessage* (с *GameEvent* в дескрипторе). Пример приведен в листинге 2.16.

```
{
  "MessageType": "EVENT",
  "MessageId": 12,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
    "EventId": 2048,
      "EventType": "BET_ACCEPTED",
        "NewStack": 30000
  }
}
```

Листинг 2.16: Пример *EventMessage*, уведомляющий о действии игрока

Со второго круга торгов в каждом раунде на стол в открытую выкладываются карты: на втором три карты, на третьем одна и на четвертом еще одна. Таким образом к четвертому кругу торгов на столе в открытую лежат пять карт. При открытии карт игровой сервер рассылает подключенным клиентам *EventMessage* (с *GameEvent* в дескрипторе) пример которого приведен в листинге 2.17. На данном листинге приведен пример открытых карт на втором круге торгов.

```
{
  "MessageType": "EVENT",
  "MessageId": 12,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
    "EventId": 564,
      "EventType": "CARDS_ON_TABLE",
      "PlayingCardsList": [
      {
        "CardSuit": "DIAMONDS",
        "Index": "10"
      },
      {
        "CardSuit": "DIAMONDS",
        "Index": "ACE"
      },
```

```
      {
        "CardSuit": "DIAMONDS",
        "Index": "KING"
      }]
  }
}
```

Листинг 2.17: Пример *EventMessage*, уведомляющий о действии игрока

Далее круги торгов повторяются аналогично описанию выше. На втором и последующих кругах торгов раунда первому игроку доступны действия: 'FOLD', 'CALL' на сумму большого блайнда, а также 'RAISE' (если ему позволяет текущий размер депозита).

Если размер депозита игрока меньше большого блайнда или текущей ставки, то ему доступно только действе 'RAISE' с коэффициентом 'ALL-IN'.

После принятия ставок на четвертом круге торгов игровой сервер выявляет самую сильную комбинацию из пяти карт среди тех, которые могут собрать игроки из карт на столе и закрытых карт игроков. Игрок, собравший эту комбинацию, является победителем раунда и забирает банк. Если таких игроков несколько, банк делится поровну между победителями. Игровой сервер рассылает подключенным к комнате клиентам результаты раунда в *EventMessage* (с *GameEvent* в дескрипторе), листинг 2.18.

```
{
  "MessageType": "EVENT",
  "MessageId": 12,
    "EventType": "GAME-EVENT",
  "EventDescriptor": {
      "EventId": 1984,
      "EventType": "WINNER_RESULT",
      "ClosedCards": [
        [
                {
                    "CardSuit": "SPADES",
                  "Index": "5"
                },
              {
                    "CardSuit": "CLUBS",
                  "Index": "ACE"
                }
            ],
        [
                {
                    "CardSuit": "DIAMONDS",
                  "Index": "JACK"
                },
              {
                    "CardSuit": "DIAMONDS",
                  "Index": "QUEEN"
                }
```

```
            ],
        [
                {
                    "CardSuit": "CLUBS",
                  "Index": "2"
                },
              {
                    "CardSuit": "HEARTS",
                  "Index": "10"
                }
            ]
        ],
      "PlayerUids": [
        "0c77ed3c-c987-4266-9579-63bd552131e6",
        "a0c85419-fa1c-449d-b828-1760a1be0c1a",
        "e63e2371-94f8-4bc7-8835-9602385deb14"],
      "WinnerUids": [
        "a0c85419-fa1c-449d-b828-1760a1be0c1a"],
      "BestCombinations":[
        [
                {
                    "CardSuit": "DIAMONDS",
                  "Index": "ACE"
                },
              {
                    "CardSuit": "DIAMONDS",
                  "Index": "KING"
                },
              {
                    "CardSuit": "DIAMONDS",
                  "Index": "QUEEN"
                },
              {
                    "CardSuit": "DIAMONDS",
                  "Index": "JACK"
                },
              {
                    "CardSuit": "DIAMONDS",
                  "Index": "10"
                }
            ]
      ],
      "BestCombName": "РОЯЛ-ФЛЕШ",
      "WinnerDeposits": [14856],
      "NewStack": 0
  }
}
```


Листинг 2.18: Пример *EventMessage*, уведомляющий о действии игрока

После подведения итогов раунда игроки, оказавшиеся с нулевыми депозитами, покидают комнату: игровой сервер рассылает всем клиентам, подключенным к комнате, сообщения (аналогичные тому, что описано в листинге 2.5) о покидании комнаты проигравшими игроками, затем сервер закрывает WebSocket-соединения с их клиентами и обновляет статистику по количеству игр и побед.

Также стоит отметить, что на стадии игрового процесса любой игрок тоже может покинуть комнату, в этом случае клиент пришлет аналогичное сообщение (см. листинг 2.4), а игровой сервер отправит остальным клиентам сообщение аналогичное тому, что приведено в листинге 2.5. Но при этом если игрок покидает комнату на стадии игрового процесса, то ему засчитывается техническое поражение и это отражается в его статистике игр и побед. Если покинувший комнату игрок был дилером, то статус дилера передается предыдущему игроку в очередности игроков (см. таблицу 2.8 и описание к ней). Если черед хода принадлежал игроку, покинувшему комнату, то ход передается следующему игроку в очереди (см. таблицу 2.13 и описание к ней).

Когда в комнате остается подключен один клиент, сервер ему присылает сообщение об изменении статуса комнаты на 'DISSOLUTION' (см. листинг 2.6), закрывает WebSocket-соединение, обновляет статистику побед и поражений игрока и изменяет состояние комнаты на 'DISSOLUTION'.

Расформирование комнаты

Игровая комната, имеющая статус 'DISSOLUTION' готова к удалению. Сервер отслеживает такие комнаты и удаляет.

2.3.4 Обработка потери соединения

В разрабатываемом протоколе должна осуществляться поддержка восстановления соединения с сервером, если вдруг по какой-либо причине, WebSocket-соединение было потеряно. Схема сценария потери соединения с сервером и его восстановление представлена ниже, на рисунке 2.9.

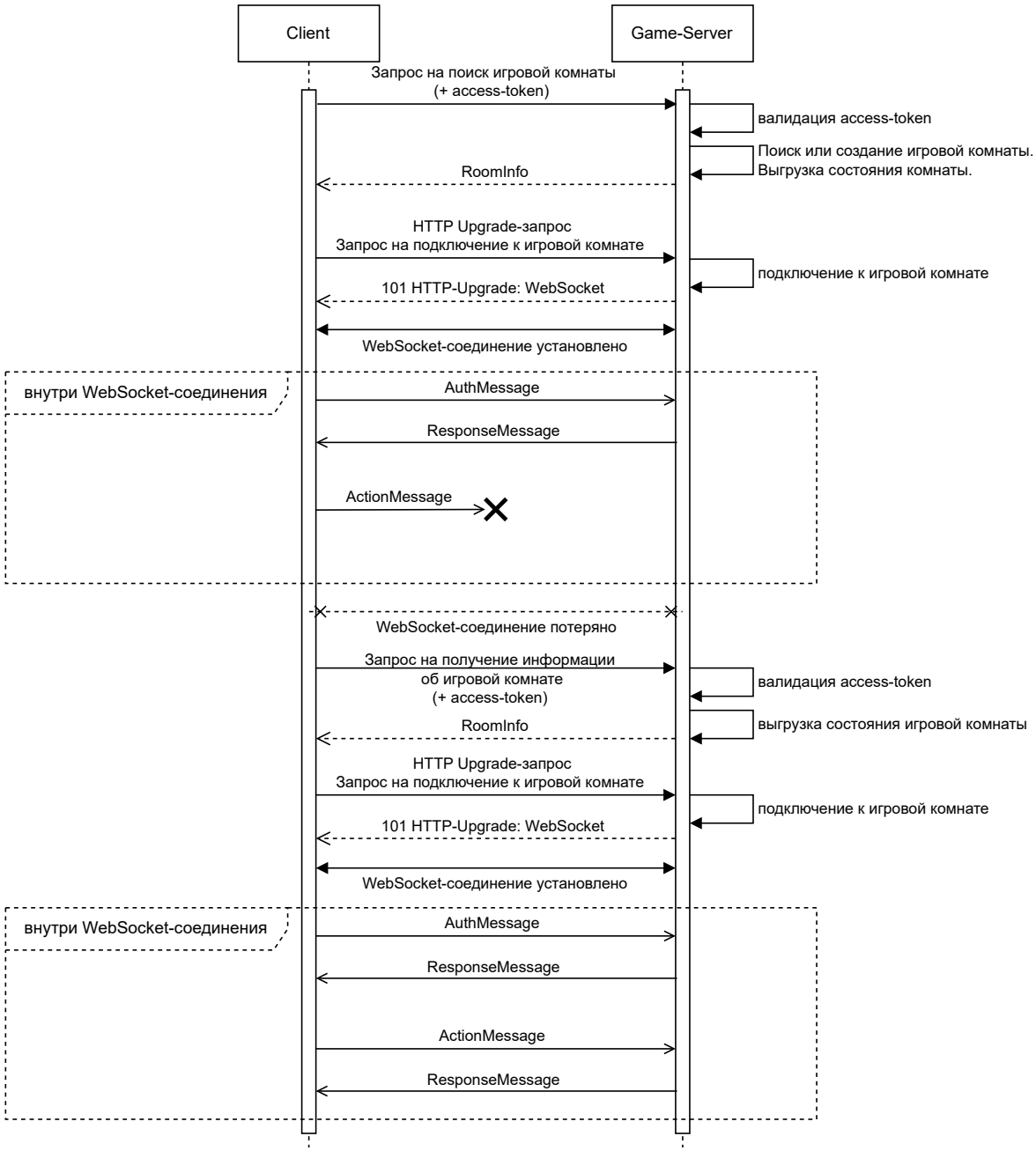

Рисунок 2.8: Потеря и восстановление соединения во время игрового процесса

Если во время игры было потеряно WebSocket-соединение с сервером, (при этом на стороне клиента должны были остаться данные *RoomUid*, и *AccessToken* игрока), то на сервер сначала отправляется запрос о получении состояние комнаты по ее идентификатору (GET */poker/v1/rooms/{roomUid}*). В качестве результата будет получен объект *RoomInfo*. Затем происходит попытка подключения к комнате, аналогичная той, что описана в разделе 2.3.2 «Поиск игровой комнаты и подключение» (см. описание к таблицам 2.10–2.17). Если соединение было потеряно на время, за которое прошел черед хода игрока или наступил его ход (и время его завершения подходит к концу), а соединение так и не было восстановлено, то ход игрока автоматически совершается сервером (без взимания игровой валюты ('FOLD'/'CHECK'), за исключением наличия на руках игрока комбинации 'РОЯЛ-ФЛЕШ' (в данном случае автоматически выполняется 'ALL-IN').

2.3.5 Завершение сессии

При выходе из приложения (или клике по кнопке выхода из аккаунта) клиент отправляет HTTP-запрос (*DELETE /poker/v1/oauth/revoke headers={ 'Authorization': RefreshToken}*). Возможные ответы на запрос о завершении сессии представлены в таблице 2.25.

Таблица 2.25: Ответ на запрос о завершении сессии

Статус код	**Ответ**	**Описание**
204		Успешное завершение сессии
500	ErrorResponse	Внутренняя ошибка сервера

3 Технологическая часть

3.1 Выбор средств программной реализации

В ходе разработки было решено использовать следующие языки программирования:

— *Golang* – для разработки серверной части приложения [11];

— *Python* – для разработки клиентской части приложения [12].

Выбор языка Golang обусловлен следующими факторами:

— Golang представляет собой статически типизированный язык программирования, обеспечивающий высокую степень надежности и безопасности кода за счет строгого контроля типов;

— Golang поддерживает встроенные механизмы параллелизма через goroutines и каналы, что критично при разработке высоко нагруженных систем;

— Golang включает обширный набор стандартных библиотек, среди которых присутствуют модули для работы с сетевыми взаимодействиями, различными форматами данных, необходимыми для реализации протоколов обмена сообщений.

Python был выбран исходя из следующих соображений:

— предоставляет поддержку разного рода приложений (в рамках данного проекта классическое Desktop-приложение);

— можно создавать кроссплатформенные приложения;

— поддерживает большое количество библиотек и фреймворков для реализации графического пользовательского интерфейса на высоком уровне.

Также серверная часть приложения развернута на арендованной машине у *VK Cloud* [13] с использованием *Doker* [14]. Для хранения информации о пользователях используется СУБД *PostgreSQL* [15].

3.2 Интерфейс приложения

Ниже, на рисунках 3.1 - 3.4 представлен графический пользовательский интерфейс программы, реализующей разработанный протокол. Также в рамках данной работы была разработана оригинальная авторская колода на 52 карты.

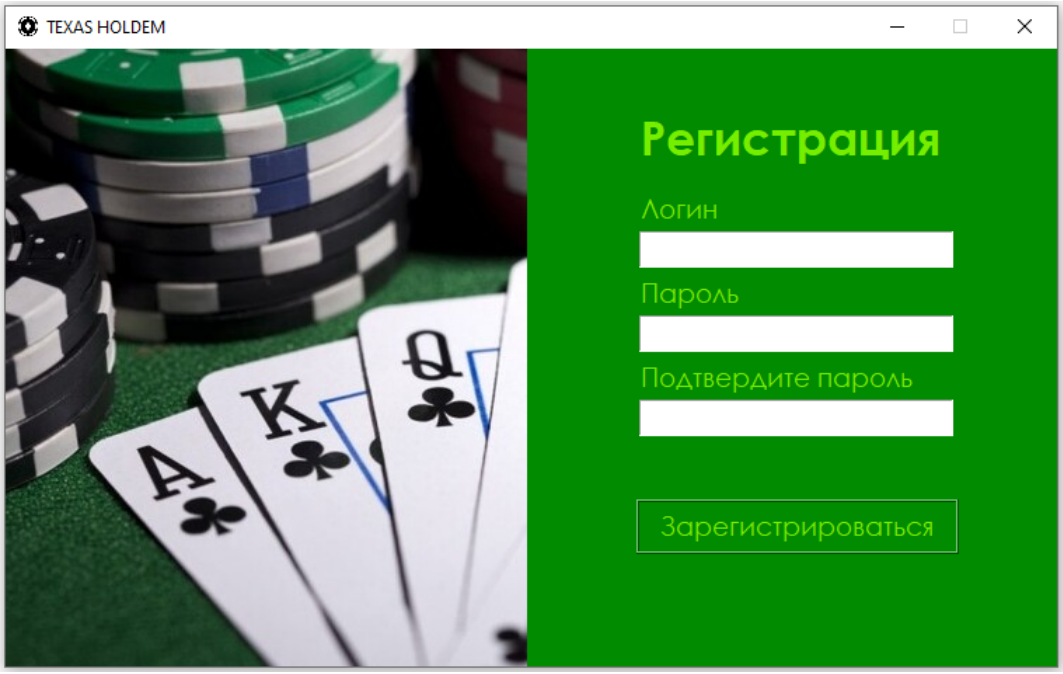

Рисунок 3.1: Окно регистрации

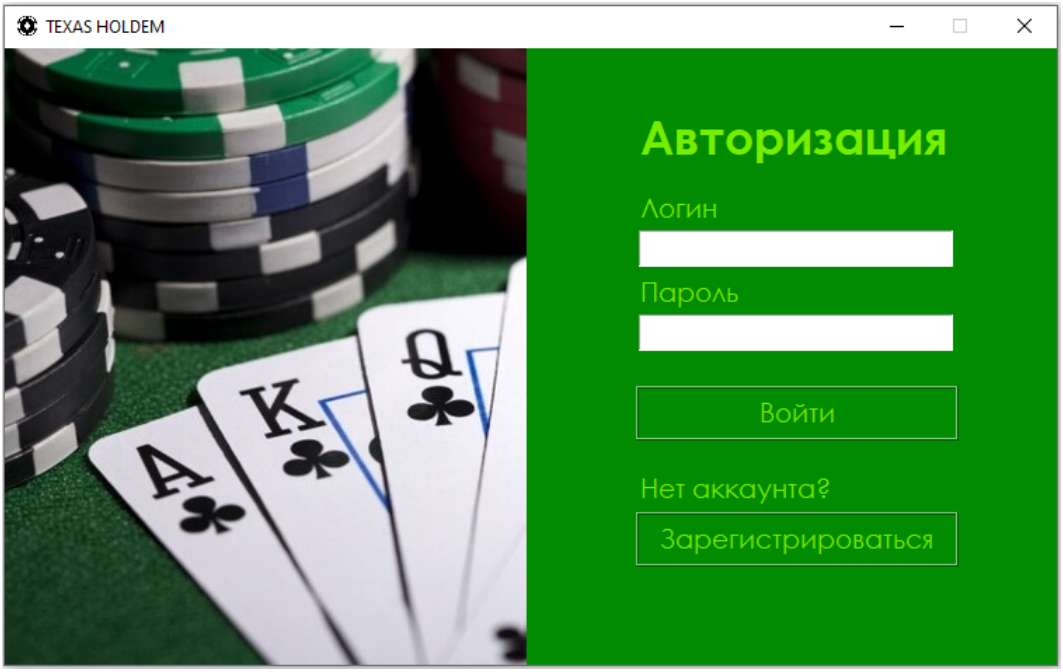

Рисунок 3.2: Окно авторизации

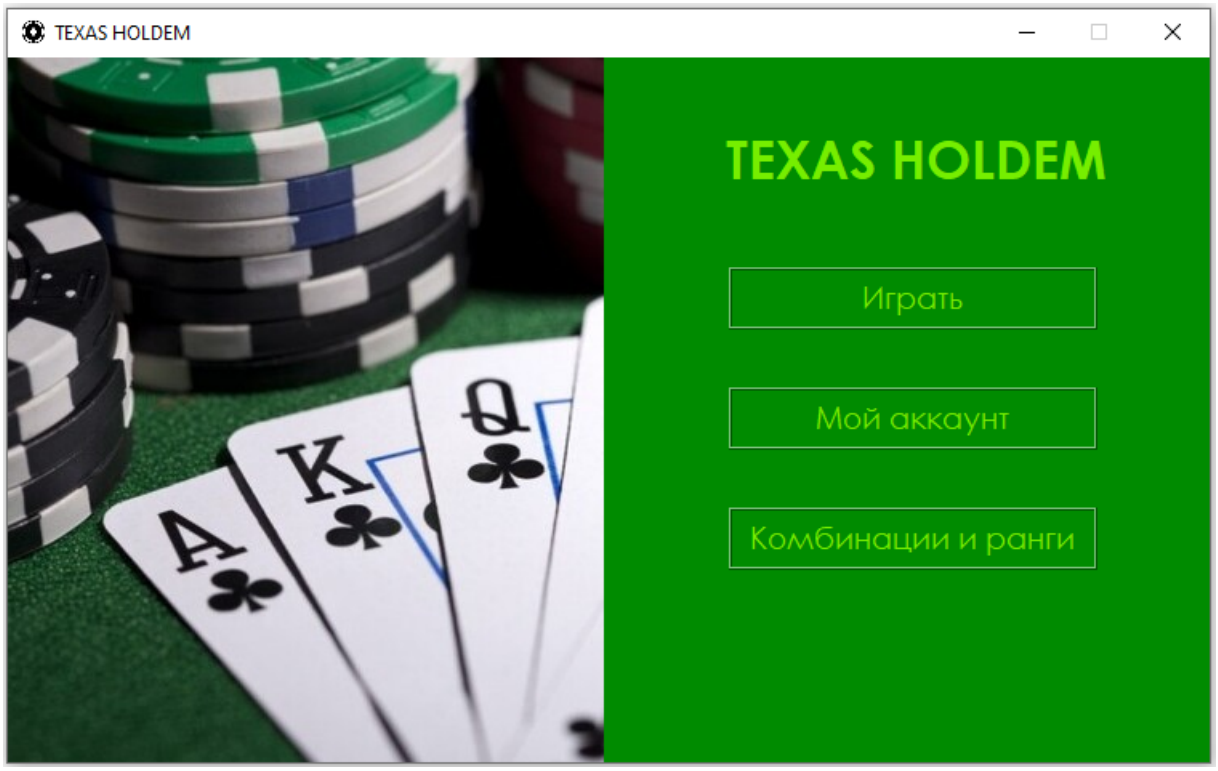

Рисунок 3.3: Основное меню

Рисунок 3.4: Игровой стол

3.3 Тестирование приложения

В рамках тестирования серверной части приложения были проведены тесты по корректности передачи и обработки информации в соответствии с разработанным протоколом. Подробнее информация о тестировании представлена ниже, в таблице 3.1.

Таблица 3.1: Тестирование клиентского приложения

Тесты	**Статус**
Формирование сообщений, отправляемых клиенту в соответствии с протоколом	Корректно
Формирование запросов к базе данных с информацией о пользователях	Корректно
Обработка сообщений, содержащих информацию о действиях игроков	Корректно
Обработка внутренней информации (ведение игрового процесса партии в соответствии с протоколом) и отправка соответствующих сообщений клиенту	Корректно

В рамках тестирования клиентской части приложения, разработанного для реализации протокола для онлайн-игры в покер были проведены несколько отладочных партий игры, в которых проверялась корректность отображения обновляющейся информации в графическом пользовательском интерфейсе по ходу партии, формирования запросов для посылки на сервер, обработки ответов от сервера и проч. Подробнее информация о тестировании представлена ниже, в таблице 3.2.

Таблица 3.2: Тестирование клиентского приложения

Тесты	**Статус**
Формирование запросов с учетом данных, введенных пользователем в соответствии с протоколом	Корректно
Обработка информации ответов от сервера и ее отображение	Корректно
Обработка обновленной информации в рамках партии (ставки, открытие карт на столе, действия других игроков и т. д.), приходящей в сообщениях от сервера и ее отображение	Корректно
Отображение локально хранящихся элементов (изображений карт, рангов) в соответствии с информацией, содержащихся в сообщениях от сервера	Корректно
Предоставление или блокировка возможности совершения игроками тех, или иных действий, порождающих запросы серверу в данный момент времени в соответствии с протоколом	Корректно

Во всех рассмотренных случаях приложение отработало корректно, в соответствии с разработанным протоколом.

ЗАКЛЮЧЕНИЕ

В результате проведенной работы был разработан протокол онлайн-игры «Покер» на основе правил Техасского Холдема.

Решены следующие задачи:

— проведен обзор предметной области (рассмотрены основные разновидности покера и их правила игры);

— выделена целевая аудитория использования протокола совместно с предлагаемым программным обеспечением;

— определены участники игрового процесса и их функции;

— определен способы взаимодействия игроков в рамках партии и составлен набор соответствующих сообщений протокола и их содержание;

— реализовано и протестировано клиент-серверное приложение, реализующее разработанный протокол.

Цель курсовой работы была достигнута. Все поставленные задачи были выполнены в полном объеме.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Большая российская энциклопедия. ПОКЕР / Линдер В. И., Кравец С. Л. [Электронный ресурс]. Режим доступа: https://old.bigenc.ru/sport/text/3151414 (Дата обращения: 15.12.2024).

2. Тихонова В. Л. Специфика развития игровой онлайн-индустрии. // Нефтегазовые технологии и экологическая безопасность – 2020. – №1(69) – С. 76-79.

3. Poker Club Management. Комбинации карт в покере [Электронный ресурс]. – 2023. – Режим доступа: https://pokercm.com/baza-znaniy/pokernye-kombinatsii/(Дата обращения: 20.12.2024).

4. Poker Club Management. Все популярные виды покера и правила игры в них [Электронный ресурс]. – 2023. – Режим доступа: https://pokercm.com/baza-znaniy/vidy-pokera-i-ih-pravila/(Дата обращения: 20.12.2024).

5. RFC6455. The WebSocket Protocol [Электронный ресурс]. – 2011. – Режим доступа: https://datatracker.ietf.org/doc/html/rfc6455(Дата обращения: 25.12.2024).

6. HTTP Documentation [Электронный ресурс]. Режим доступа: https://httpwg.org/specs/(Дата обращения: 25.12.2024).

7. Что такое OpenID Connect (OIDC)? [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/security/business/security-101/what-is-openid-connect-oidc(Дата обращения: 25.12.2024).

8. Что такое OAuth? [Электронный ресурс]. Режим доступа: https://www.microsoft.com/ru-ru/security/business/security-101/what-is-oauth(Дата обращения: 25.12.2024).

9. Академия Покера. Дилер в покере [Электронный ресурс]. – 2017. – Режим доступа: https://academypoker.ru/terms/1900-diler-v-pokere.html(Дата обращения: 25.12.2024).

10. JSON [Электронный ресурс]. Режим доступа: https://www.json.org/json-en.html(Дата обращения: 25.12.2024).

11. Golang. Официальная документация. [Электронный ресурс]. Режим доступа: https://pkg.go.dev/std (Дата обращения: 26.12.2024).

12. Python. Официальная документация. [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (Дата обращения: 26.12.2024).

13. VK Cloud. Официальная документация. [Электронный ресурс]. Режим доступа: https://cloud.vk.com/docs/ (Дата обращения: 26.12.2024).

14. Docker. Официальная документация. [Электронный ресурс]. Режим доступа: https://docs.docker.com/ (Дата обращения: 26.12.2024).

15. PostgreSQL. Официальная документация. [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/ (Дата обращения: 26.12.2024).

ПРИЛОЖЕНИЕ А
Демонстрация работы протокола (логи WireShark)

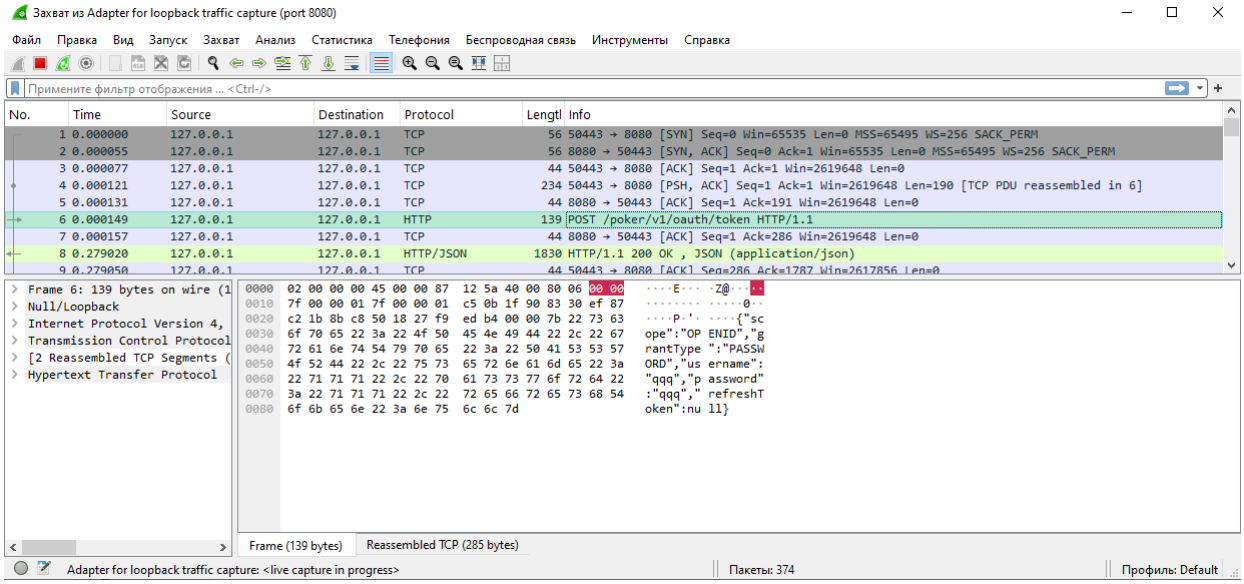

Рисунок 3.1: HTTP-запрос на авторизацию

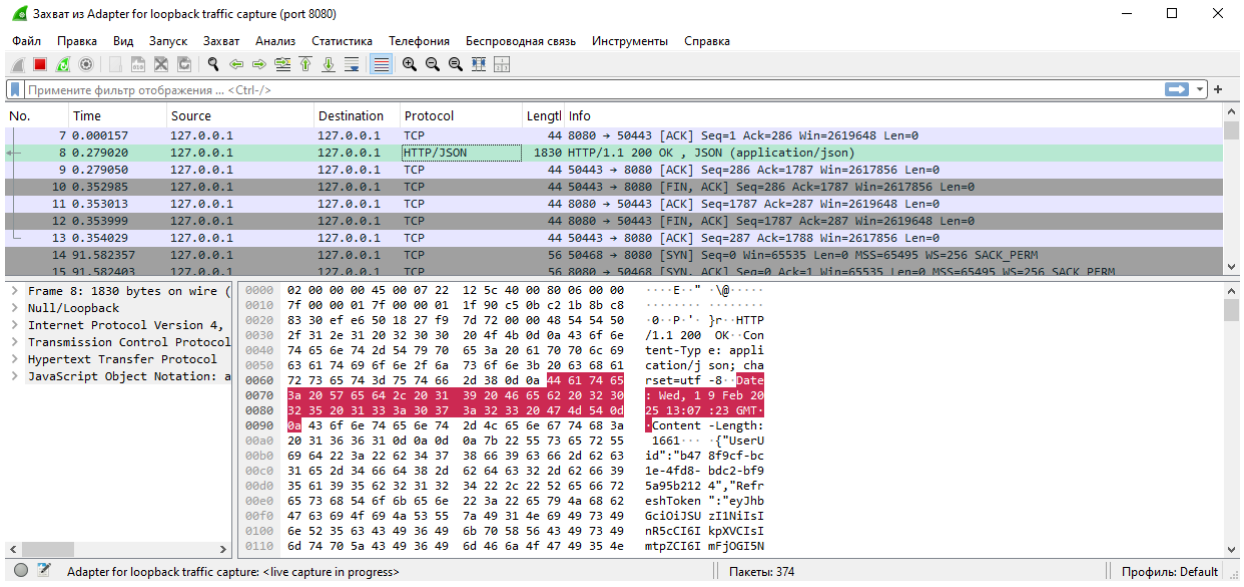

Рисунок 3.2: Ответ на авторизацию

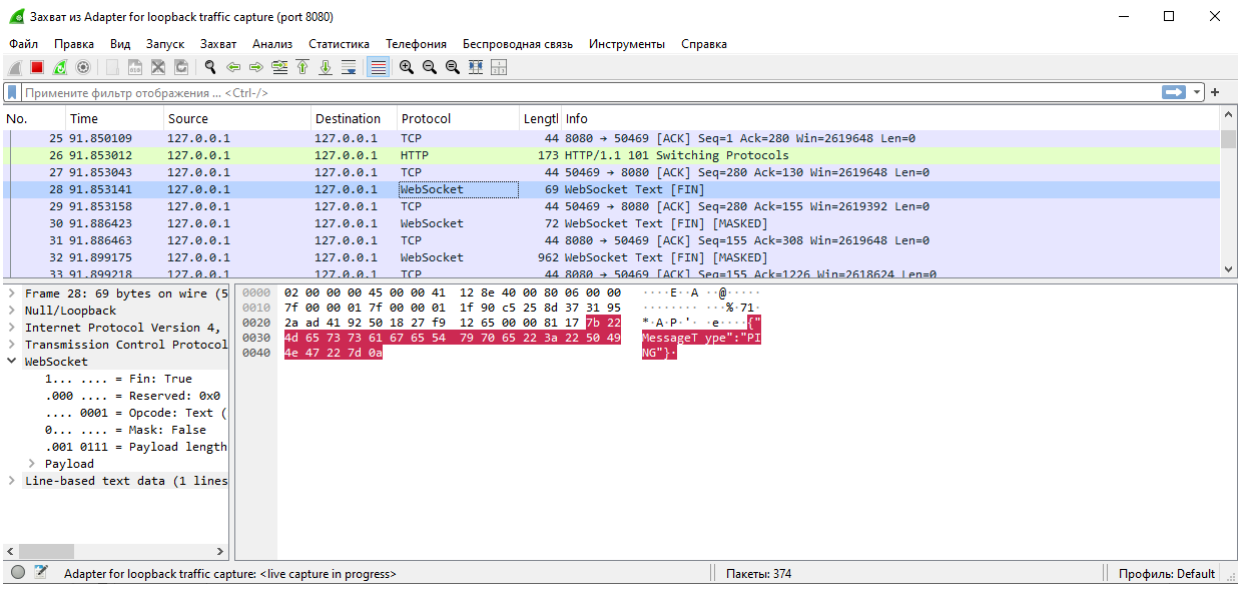

Рисунок 3.3: WebSocket PING

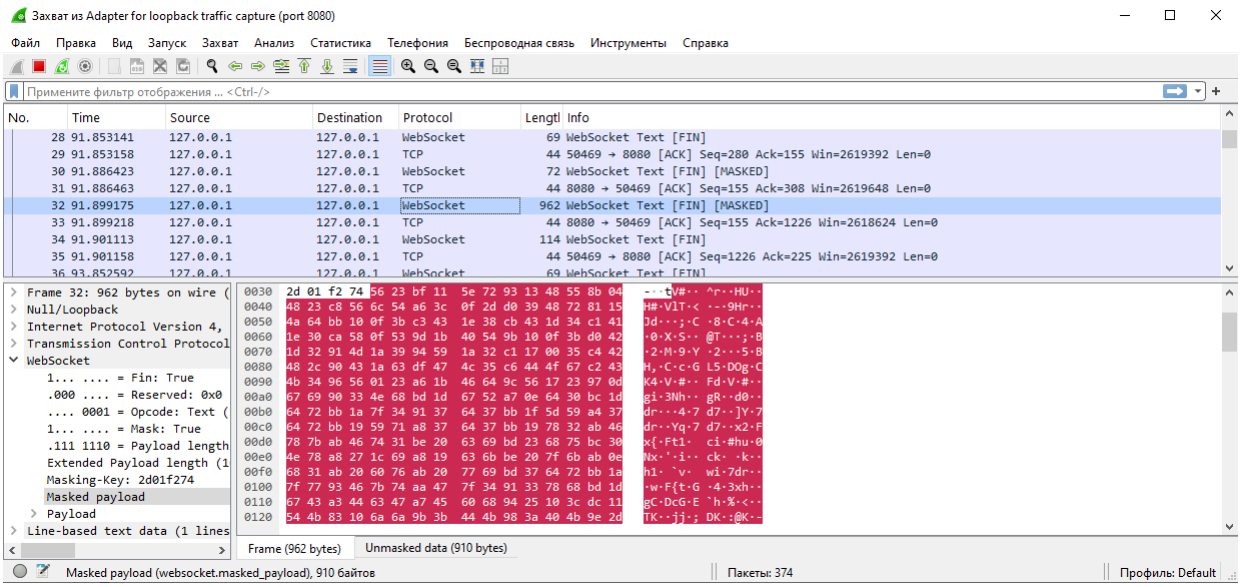

Рисунок 3.4: WebSocket message

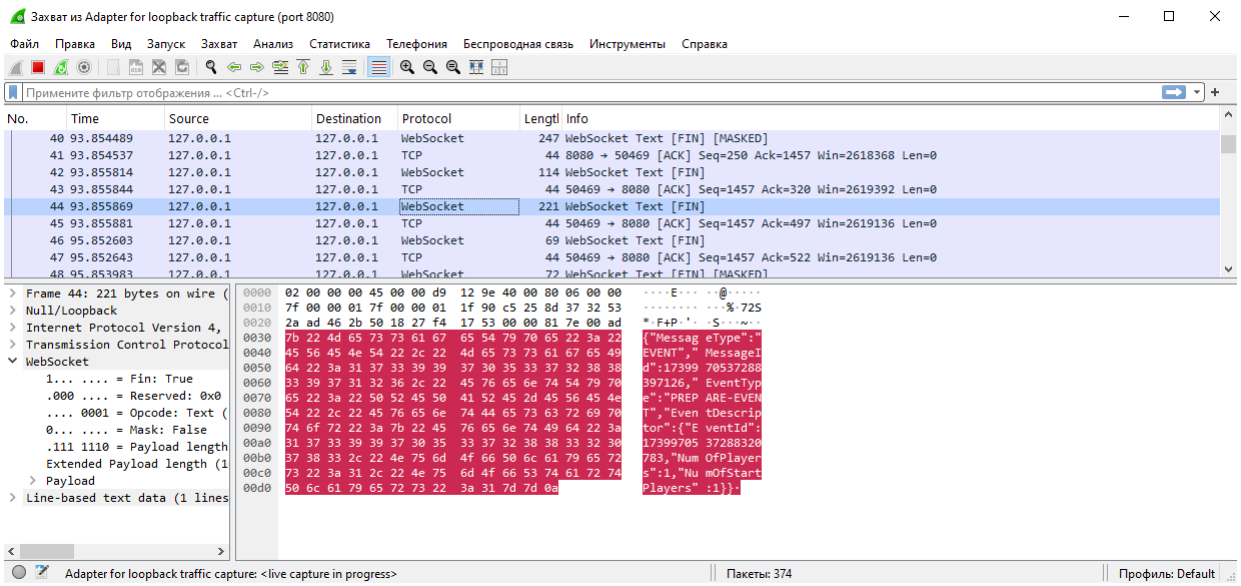

Рисунок 3.5: WebSocket PREPARE-EVENT

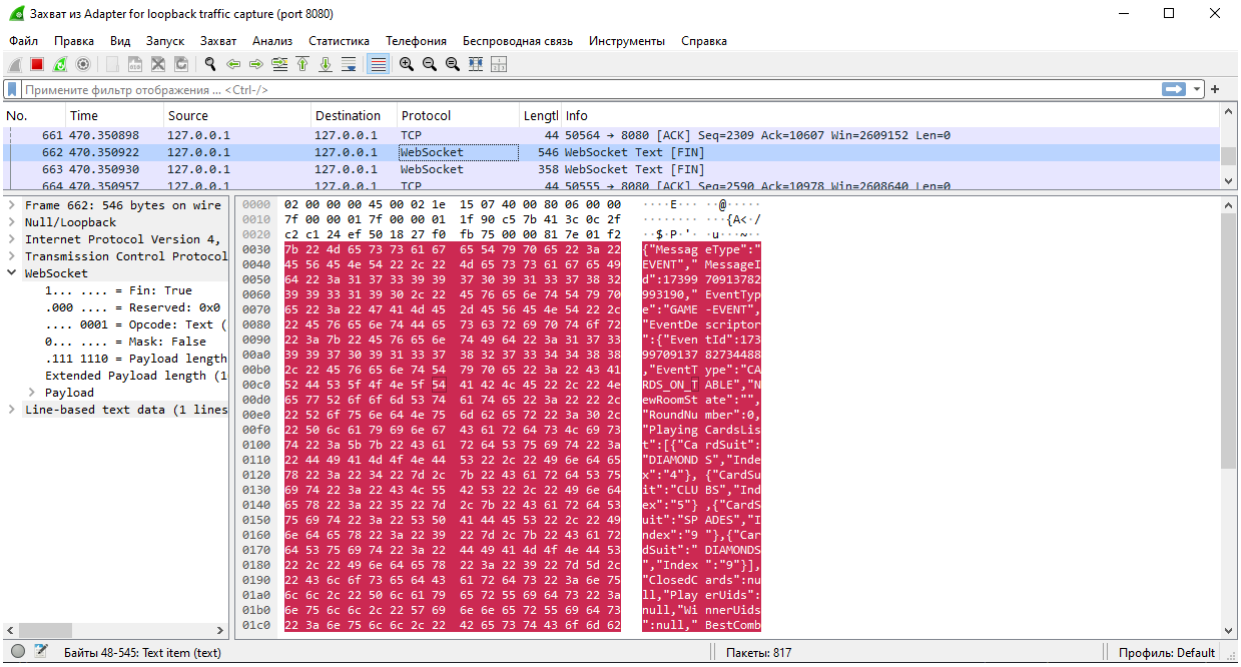

Рисунок 3.6: WebSocket CARDS_ON_TABLE

ПРИЛОЖЕНИЕ Б

Демонстрация работы протокола (логи WebSoket)

```
—— request header ——
GET /poker/v1/rooms-ws/b7989deb-0ad6-4ad4-ace9-69f23dac1e59?uid=69e3f858-80aa-456b-8614-562c17fbc816 HT
TP/1.1
Upgrade: websocket
Host: localhost:8080
Origin: http://localhost:8080
Sec-WebSocket-Key: RJFRCbaKqNYrptcj99QqVQ==
Sec-WebSocket-Version: 13
Connection: Upgrade



————————————————
—— response header ——
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: OEfgaaLQyJL3oXsq8GL+sLmkTzE=
————————————————
Websocket connected
###### opened ######
++Rcv raw: b'\x81\x17{"MessageType":"PING"}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"PING"}\n'
++Sent raw: b'\x81\x96f#\x88+\x1d\x01\xc5N\x15P\xe9L\x03w\xf1[\x03\x01\xb2\t6l\xc61D^'
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"PONG"}'
++Sent raw: b"\x81\xfe\x03\x8e\xd9\x1d>\x18\xa2?s}\xaan_\x7f\xbcIGh\xbc?\x04:\x98HjP\xfb1\x1cU\xbcnMy\x
bexw|\xfb'\x0f/\xea$\x07.\xe1,\x0c-\xeb-\x0b4\xfbOQw\xb4HW|\xfb'\x1cz\xee$\x06!\xbdx\\5\xe9|Z.\xf4)_|\x
ed0_{\xbc$\x13.\xe0{\x0c+\xbd|])\xbc(\x07:\xf5?jw\xb2xP:\xe3?[a\x93u\\_\xbatqq\x93Nkb\x90,pq\x90nwv\x8b
(][\x90+ws\xa9Eh[\x90nwu\xadmd[\x90+wu\x8c.g*\x8cgg*\x80-rL\x97uqO\x9cip\\\xbaddK\xe8udu\x97wrL\x8bwgb\
x9c-gL\x94jgL\x83uq[\x90nwv\x8bk_*\x8fhf+\x8b(]_\x8ctqq\x93_o(\x97[k)\x94tXI\xe4 \x10}\xa0WO|\x9evWW\xb
0TDV\xa3qUB\xb4[TU\x9a-\x0eU\x9d\\IT\x8dOTW\x9dHJW\x8d\\\nU\x8a-IV\x9dXFW\x8dXGB\x9dX\x0eB\x8dPWT\x9aW\
x0f{\xebKG@\xeaKNB\x9aT\x08Q\xb3D\x0bB\x8dSSW\x9dH\nT\x8dzIA\x8eXJV\x9dH\x0cA\xb0-\nV\xb3X\x0eT\x8dH\x0
cU\xb4PFV\xebGWA\xa3zFV\xb0TMQ\xb4qD{\xa0T\x08Q\xb4u\x0e|\x91\\\x08T\xa0$NB\x9eKK|\x9eq\x0e}\x8a,I{\xb4
$\x0cy\x8eOR{\xb3r\nU\x9dvIT\xa0TMQ\xb4qV|\x9aT\x08U\x8d~DW\x8dv\x0cW\x9dXGU\xa0jWB\x81uIQ\xb3rFV\xa3P\
x0bW\x8d~FV\xa3TDT\x9aWUB\x81GNA\xebKXy\x8eLWW\xb0T\x0eB\xb3_UV\x9dL\x0eA\xa0-FB\x9dH\x0bT\x8dOUB\x8dLJ
A\x8eWTA\xa0,VB\xb3[UW\x9dz\x0fU\xb3X\x0cV\xa3\\W~\x88 \x036\xa8~l\\\xacN]Z\xafvot\xa8h{J\x8a|Mi\xe0Grl
\x94\\\x0fn\xe0GV5\x8bvd}\xe1IF\\\xefpw/\xb2Ts]\x9axXa\xb3Wj|\x9bWFK\xa9lVn\x8dkUA\xb8hzB\x86Oli\xeexnm
\x9dN_b\x86OUj\xaaId\x7f\xa0n\tA\xb4vmP\x89yFR\xa0ktw\xb2DT^\xa3+d]\xe1guz\x9crvO\xe1~kL\x89$\x13z\x9fY
\x0e]\xb8lLP\xe9wny\x97,\x07l\x9eEw)\xa3Psb\x8d^|I\x9apf5\xa3zpZ\xebuvr\xb6s\x08Z\x8c+\x13t\xe9q_!\xb1y
gb\x9axQp\xe9hdk\xaaLhy\xbcJDs\xb5\\HM\x90~Z \xf4)\tH\x92Vqw\x9c%LS\xe1t]r\xacKio\xac0}(\xb1mQi\xa9NZG\
xe9Poo\x8du[b\x8dv\x06H\x8a.\x13w\xaevfP\x90Xrw\xa8iP[\xb0UOZ\x8cv\x08O\xaaNjQ\x86so[\xb2\\\tK\x96gmR\x
9eTaY\x8aRd}\xbb,X|\xa3Zi)\x95YKN\x9fB\x0bU\xb5eKG\xa9SZl\x8d|_M\x8e^G]\x9fX\x0es\x9dj\x1c4\xfbQ_k\xadX
H}\xb7iw|\xfb'\x0f/\xea$\x07.\xe1,\x0c,\xee$\x0e)\xea+\x0c.\xeb‘"
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"AUTH","MessageId":1739968125205,"RoomUid":"b7989d
eb-0ad6-4ad4-ace9-69f23dac1e59","Token":"eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCIsImtpZCI6ImU3Y2UzY2Y0LTNhOW
EtNDcyZS1hZmNjLTRjYzE0YTMwYTZhOCIsInRva2VuX3R5cGUiOiJBQ0NFU1MifQ==.eyJqdGkiOiIzNzlkZmFjMC00MDAwLTRjODUt
OTA4MS0wNDExOTEyZDE0ZTMiLCJ1c2VyX3VpZCI6IjY5ZTNmODU4LTgwYWEtNDU2Yi04NjE0LTU2MmMxN2ZiYzgxNiIsImlzcyI6Imh
0dHA6Ly9pZGVudGl0eS1wcm92aWRlcjo4MDkwLyIsImlhdCI6MTczOTk2ODEyMywiZXhwIjoxNzM5OTcxNzIzLCJkZXZpY2VfaWQiOi
I0ZjBkNDQ0Yy0xZDU5LTRkZTQtYWJjYy1hZjFkODg1MjE2NzAifQ==.qcRDuScBvkQlquERSasq9ZLtMA1v9Zh-RkZe8TxD6mI7kIME
CefyjJTdBJxSpqhvTvkYauDZ_RRq7ePuDSaz_RkrsTZgys7YmkSHPdxJyvJokYjFz6ZE8zKbEoHW8cUTP9-bFD0EaqrH0jPaN19tGXI
1zMMzTCBQCmX-zgNB2hHjon6BU6-l0la9hdYzCeoh0uZssQVaeWzklAvUIcd8-47PKKOoE8rK8icjuVWwu-C0hpoqpSd_0MQwThezTk
8PS3-owkXHIELoqtnCiHqBUk6WsSTI_nQCkA7SOzSJGI_ASOZeb1fdzGW1LDuVF_5Mlxu_pNdtTaaUWCyEFE0kDw","LastEventId"
:1739968124790136262}'
++Rcv raw: b'\x81D{"MessageType":"ACK","AckMessageId":1739968125205,"StatusCode":200}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"ACK","AckMessageId":1739968125205,"StatusCode":200
}\n'
++Rcv raw: b'\x81\x17{"MessageType":"PING"}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"PING"}\n'
++Sent raw: b'\x81\x96\xee\xe2\x0c\xcb\x95\xc0A\xae\x9d\x91m\xac\x8b\xb6u\xbb\x8b\xc06\xe9\xbe\xadB\x8c
\xcc\x9f'
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"PONG"}'
++Sent raw: b'\x81\xfe\x00\xc3%k\xf5\xac^I\xb8\xc9V\x18\x94\xcb@?\x8c\xdc@I\xcf\x8es$\xa1\xe9\x07G\xd7\
xe1@\x18\x86\xcdB\x0e\xbc\xc8\x07Q\xc4\x9b\x16R\xcc\x9a\x1dZ\xc7\x9b\x14]\xc4\x80\x079\x9a\xc3H>\x9c\xc
8\x07Q\xd7\xce\x12R\xcd\x95A\x0e\x97\x81\x15\n\x91\x9a\x08_\x94\xc8\x11F\x94\xcf@R\xd8\x9a\x1c\r\xc7\x9
fA\n\x96\x9d@^\xcc\x8e\tI\xa0\xdf@\x19\xa0\xc5AI\xcf\x8e\x13R\x90\x9fCS\xc0\x94\x08S\xc5\xcdDF\xc1\x99\
x13\t\xd8\x94\x13Z\xc1\x81\x10]\xc7\xcf\x14\\\x93\xceFS\xc4\x9a\x07G\xd7\xedF\x1f\x9c\xc3K?\x8c\xdc@I\x
cf\xc2P\x07\x99\x80\x07(\x9a\xc9CI\xcf\xc2P\x07\x99\x80\x07=\x9a\xd8@?\x8c\xdc@I\xcf\x8ev?\xb4\xfeqI\x8
8'
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"VOTE","MessageId":1739968127161,"RoomUid":"b7989d
eb-0ad6-4ad4-ace9-69f23dac1e59","UserUid":"69e3f858-80aa-456b-8614-562c17fbc816","ActionType":null,"Coe
f":null,"VoteType":"START"}'
```

```
++Rcv raw: b'\x81D{"MessageType":"ACK","AckMessageId":1739968127161,"StatusCode":200}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"ACK","AckMessageId":1739968127161,"StatusCode":200
}\n'
++Rcv raw: b'\x81~\x00\xad{"MessageType":"EVENT","MessageId":1739968127167958931,"EventType":"PREPARE-E
VENT","EventDescriptor":{"EventId":1739968127167858831,"NumOfPlayers":2,"NumOfStartPlayers":2}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127167958931,"EventType"
:"PREPARE-EVENT","EventDescriptor":{"EventId":1739968127167858831,"NumOfPlayers":2,"NumOfStartPlayers":
2}}\n'
++Rcv raw: b'\x81~\x01g{"MessageType":"EVENT","MessageId":1739968127168494532,"EventType":"GAME-EVENT",
"EventDescriptor":{"EventId":1739968127168486832,"EventType":"ROOM_STATE_UPDATE","NewRoomState":"GAMING
","RoundNumber":0,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestC
ombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168494532,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968127168486832,"EventType":"ROOM_STATE_UPDATE","NewRoom
State":"GAMING","RoundNumber":0,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUid
s":null,"BestCombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv raw: b'\x81~\x01Y{"MessageType":"EVENT","MessageId":1739968127168715133,"EventType":"GAME-EVENT",
"EventDescriptor":{"EventId":1739968127168527732,"EventType":"NEW_ROUND","NewRoomState":"","RoundNumber
":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinations":n
ull,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168715133,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968127168527732,"EventType":"NEW_ROUND","NewRoomState":"
","RoundNumber":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestC
ombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv raw: b'\x81~\x010{"MessageType":"EVENT","MessageId":1739968127168785733,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968127168529532,"UserUid":"b478f9cf-bc1e-4fd8-bdc2-bf95a95b21
24","ActionType":"SET-DEALER","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"NewBet":0
,"NewDeposit":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168785733,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968127168529532,"UserUid":"b478f9cf-bc1e-4fd8-b
dc2-bf95a95b2124","ActionType":"SET-DEALER","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming
":0,"NewBet":0,"NewDeposit":0}}\n'
++Rcv raw: b'\x81~\x01_{"MessageType":"EVENT","MessageId":1739968127168804833,"EventType":"GAME-EVENT",
"EventDescriptor":{"EventId":1739968127168532232,"EventType":"NEW_TRADE_ROUND","NewRoomState":"","Round
Number":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinati
ons":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168804833,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968127168532232,"EventType":"NEW_TRADE_ROUND","NewRoomSt
ate":"","RoundNumber":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,
"BestCombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv raw: b'\x81~\x017{"MessageType":"EVENT","MessageId":1739968127168819433,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968127168534032,"UserUid":"69e3f858-80aa-456b-8614-562c17fbc8
16","ActionType":"MIN-BLIND-IN","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"NewBet"
:250,"NewDeposit":9750}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168819433,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968127168534032,"UserUid":"69e3f858-80aa-456b-8
614-562c17fbc816","ActionType":"MIN-BLIND-IN","BoutVariants":null,"BestCombName":"","TimeEndBoutOrFormi
ng":0,"NewBet":250,"NewDeposit":9750}}\n'
++Rcv raw: b'\x81~\x017{"MessageType":"EVENT","MessageId":1739968127168834533,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968127168535432,"UserUid":"b478f9cf-bc1e-4fd8-bdc2-bf95a95b21
24","ActionType":"MAX-BLIND-IN","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"NewBet"
:500,"NewDeposit":9500}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168834533,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968127168535432,"UserUid":"b478f9cf-bc1e-4fd8-b
dc2-bf95a95b2124","ActionType":"MAX-BLIND-IN","BoutVariants":null,"BestCombName":"","TimeEndBoutOrFormi
ng":0,"NewBet":500,"NewDeposit":9500}}\n'
++Rcv raw: b'\x81~\x01\xa7{"MessageType":"EVENT","MessageId":1739968127168849733,"EventType":"GAME-EVEN
T","EventDescriptor":{"EventId":1739968127168536632,"EventType":"PERSONAL_CARDS","NewRoomState":"","Rou
ndNumber":0,"PlayingCardsList":[{"CardSuit":"HEARTS","Index":"5"},{"CardSuit":"SPADES","Index":"9"}],"C
losedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinations":null,"BestCombName":"\xd0\x9f\x
d0\x90\xd0\xa0\xd0\x90","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127168849733,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968127168536632,"EventType":"PERSONAL_CARDS","NewRoomSta
te":"","RoundNumber":0,"PlayingCardsList":[{"CardSuit":"HEARTS","Index":"5"},{"CardSuit":"SPADES","Inde
x":"9"}],"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinations":null,"BestCombName":
"\xd0\x9f\xd0\x90\xd0\xa0\xd0\x90","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv raw: b'\x81~\x01\xfe{"MessageType":"EVENT","MessageId":1739968127169351134,"EventType":"PLAYER-AC
TION-EVENT","EventDescriptor":{"EventId":1739968127168593132,"UserUid":"69e3f858-80aa-456b-8614-562c17f
bc816","ActionType":"BOUT","BoutVariants":[{"VariantType":"FOLD","CallValue":0,"RaiseVariants":null},{"
VariantType":"CALL","CallValue":250,"RaiseVariants":null},{"VariantType":"RAISE","CallValue":0,"RaiseVa
riants":["X1_5","X2","ALL-IN"]}],"BestCombName":"\xd0\x9f\xd0\x90\xd0\xa0\xd0\x90","TimeEndBoutOrFormin
g":1739968187268,"NewBet":0,"NewDeposit":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968127169351134,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968127168593132,"UserUid":"69e3f858-80aa-456b-8
614-562c17fbc816","ActionType":"BOUT","BoutVariants":[{"VariantType":"FOLD","CallValue":0,"RaiseVariant
s":null},{"VariantType":"CALL","CallValue":250,"RaiseVariants":null},{"VariantType":"RAISE","CallValue"
:0,"RaiseVariants":["X1_5","X2","ALL-IN"]}],"BestCombName":"\xd0\x9f\xd0\x90\xd0\xa0\xd0\x90","TimeEndB
outOrForming":1739968187268,"NewBet":0,"NewDeposit":0}}\n'
++Rcv raw: b'\x81\x17{"MessageType":"PING"}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"PING"}\n'
++Sent raw: b'\x81\x96\xb7\xbd`\xdc\xcc\x9f-\xb9\xc4\xce\x01\xbb\xd2\xe9\x19\xac\xd2\x9fZ\xfe\xe7\xf2.\
x9b\x95\xc0'
```

```
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"PONG"}'
++Sent raw: b"\x81\xfe\x00\xc9r\x8b\xd1\xcb\t\xa9\x9c\xae\x01\xf8\xb0\xac\x17\xdf\xa8\xbb\x17\xa9\xeb\x
e95\xca\x9c\x8e_\xca\x92\x9f;\xc4\x9f\xe9^\xa9\x9c\xae\x01\xf8\xb0\xac\x17\xc2\xb5\xe9H\xba\xe6\xf8K\xb
2\xe7\xf3C\xb9\xe8\xfaD\xba\xfd\xe9 \xe4\xbe\xa6'\xe2\xb5\xe9H\xa9\xb3\xfcK\xb3\xe8\xaf\x17\xe9\xfc\xfb
\x13\xef\xe7\xe6F\xea\xb5\xff_\xea\xb2\xaeK\xa6\xe7\xf2\x14\xb9\xe2\xaf\x13\xe8\xe0\xaeG\xb2\xf3\xe7P\x
de\xa2\xae\x00\xde\xb8\xafP\xb1\xf3\xfdK\xee\xe2\xadJ\xbe\xe9\xe6J\xbb\xb0\xaa_\xbf\xe4\xfd\x10\xa6\xe9
\xfdC\xbf\xfc\xfeD\xb9\xb2\xfaE\xed\xb3\xa8J\xba\xe7\xe9^\xa9\x90\xa8\x06\xe2\xbe\xa5&\xf2\xa1\xaeP\xb1
\xf3\x883\xc7\x9d\xe9^\xa9\x92\xa4\x17\xed\xf3\xf1\x1c\xfe\xbd\xa7^\xa9\x87\xa4\x06\xee\x85\xb2\x02\xee
\xf3\xf1\x1c\xfe\xbd\xa7\x0f"
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"GAME-ACTION","MessageId":1739968129161,"RoomUid":
"b7989deb-0ad6-4ad4-ace9-69f23dac1e59","UserUid":"69e3f858-80aa-456b-8614-562c17fbc816","ActionType":"C
ALL","Coef":null,"VoteType":null}'
++Rcv raw: b'\x81D{"MessageType":"ACK","AckMessageId":1739968129161,"StatusCode":200}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"ACK","AckMessageId":1739968129161,"StatusCode":200
}\n'
++Rcv raw: b'\x81~\x01/{"MessageType":"EVENT","MessageId":1739968129164550853,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968129164452553,"UserUid":"69e3f858-80aa-456b-8614-562c17fbc8
16","ActionType":"CALL","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"NewBet":500,"Ne
wDeposit":9500}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968129164550853,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968129164452553,"UserUid":"69e3f858-80aa-456b-8
614-562c17fbc816","ActionType":"CALL","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"N
ewBet":500,"NewDeposit":9500}}\n'
++Rcv raw: b'\x81~\x016{"MessageType":"EVENT","MessageId":1739968129164577753,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968129164461753,"UserUid":"b478f9cf-bc1e-4fd8-bdc2-bf95a95b21
24","ActionType":"BOUT","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":1739968189264,"New
Bet":0,"NewDeposit":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968129164577753,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968129164461753,"UserUid":"b478f9cf-bc1e-4fd8-b
dc2-bf95a95b2124","ActionType":"BOUT","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":1739
968189264,"NewBet":0,"NewDeposit":0}}\n'
++Rcv raw: b'\x81\x17{"MessageType":"PING"}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"PING"}\n'
++Sent raw: b'\x81\x96\xf7?k\x93\x8c\x1d&\xf6\x84L\n\xf4\x92k\x12\xe3\x92\x1dQ\xb1\xa7p%\xd4\xd5B'
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"PONG"}'
++Rcv raw: b'\x81\x17{"MessageType":"PING"}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"PING"}\n'
++Sent raw: b'\x81\x96\xfd\x97\x9bG\x86\xb5\xd6"\x8e\xe4\xfa \x98\xc3\xe27\x98\xb5\xa1e\xad\xd8\xd5\x00
\xdf\xea'
++Sent decoded: fin=1 opcode=1 data=b'{"MessageType":"PONG"}'
++Rcv raw: b'\x81~\x010{"MessageType":"EVENT","MessageId":1739968134779309217,"EventType":"PLAYER-ACTIO
N-EVENT","EventDescriptor":{"EventId":1739968134779300417,"UserUid":"b478f9cf-bc1e-4fd8-bdc2-bf95a95b21
24","ActionType":"CHECK","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"NewBet":500,"N
ewDeposit":9500}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968134779309217,"EventType"
:"PLAYER-ACTION-EVENT","EventDescriptor":{"EventId":1739968134779300417,"UserUid":"b478f9cf-bc1e-4fd8-b
dc2-bf95a95b2124","ActionType":"CHECK","BoutVariants":null,"BestCombName":"","TimeEndBoutOrForming":0,"
NewBet":500,"NewDeposit":9500}}\n'
++Rcv raw: b'\x81~\x01_{"MessageType":"EVENT","MessageId":1739968134779412317,"EventType":"GAME-EVENT",
"EventDescriptor":{"EventId":1739968134779308417,"EventType":"BET_ACCEPTED","NewRoomState":"","RoundNum
ber":0,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinations
":null,"BestCombName":"","WinnerDeposits":null,"NewStack":1000}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968134779412317,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968134779308417,"EventType":"BET_ACCEPTED","NewRoomState
":"","RoundNumber":0,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"Be
stCombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":1000}}\n'
++Rcv raw: b'\x81~\x01_{"MessageType":"EVENT","MessageId":1739968134779446517,"EventType":"GAME-EVENT",
"EventDescriptor":{"EventId":1739968134779311217,"EventType":"NEW_TRADE_ROUND","NewRoomState":"","Round
Number":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinati
ons":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968134779446517,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968134779311217,"EventType":"NEW_TRADE_ROUND","NewRoomSt
ate":"","RoundNumber":1,"PlayingCardsList":null,"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,
"BestCombinations":null,"BestCombName":"","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv raw: b'\x81~\x01\xcc{"MessageType":"EVENT","MessageId":1739968134779468817,"EventType":"GAME-EVEN
T","EventDescriptor":{"EventId":1739968134779313417,"EventType":"CARDS_ON_TABLE","NewRoomState":"","Rou
ndNumber":0,"PlayingCardsList":[{"CardSuit":"HEARTS","Index":"9"},{"CardSuit":"SPADES","Index":"QUEEN"}
,{"CardSuit":"CLUBS","Index":"8"}],"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"BestCombinat
ions":null,"BestCombName":"\xd0\x9f\xd0\x90\xd0\xa0\xd0\x90","WinnerDeposits":null,"NewStack":0}}\n'
++Rcv decoded: fin=1 opcode=1 data=b'{"MessageType":"EVENT","MessageId":1739968134779468817,"EventType"
:"GAME-EVENT","EventDescriptor":{"EventId":1739968134779313417,"EventType":"CARDS_ON_TABLE","NewRoomSta
te":"","RoundNumber":0,"PlayingCardsList":[{"CardSuit":"HEARTS","Index":"9"},{"CardSuit":"SPADES","Inde
x":"QUEEN"},{"CardSuit":"CLUBS","Index":"8"}],"ClosedCards":null,"PlayerUids":null,"WinnerUids":null,"B
estCombinations":null,"BestCombName":"\xd0\x9f\xd0\x90\xd0\xa0\xd0\x90","WinnerDeposits":null,"NewStack
":0}}\n'
.............
```

Листинг 3.1: Логи Websocket с клиента

ПРИЛОЖЕНИЕ В

Код программного обеспечения

Серверная часть

```
package externalServices

import (
  "bauman-poker/config"
  "bauman-poker/schemas"
  "net/http"

  log "github.com/sirupsen/logrus"
)

type IdentityExterService struct {
  baseUrlApiv1  string
  requestSender *RequestSender
}

func NewIdentityExterService() *IdentityExterService {
  context := NewBreakerContext(config.IdentityExterBaseUrl + config.HealthCheckHandler)
  return &IdentityExterService{
    baseUrlApiv1:  config.IdentityExterBaseUrl + config.IdentityGroupName,
    requestSender: NewRequestSender(context),
  }
}

func (ies IdentityExterService) GetJWKs() []schemas.JWKey {
  url := ies.baseUrlApiv1 + config.JWKsHandler
  newReq, err := http.NewRequest(http.MethodGet, url, nil)
  if err != nil {
    log.WithError(err).Errorf("Error in IdentityExterService.GetJWKs. (err in making request). url: %s", url)
    return []schemas.JWKey{}
  }

  resp, err2 := ies.requestSender.SendRequest(newReq)
  if err2 != nil {
    log.Errorf("Error in IdentityExterService.GetJWKs. (err in sending request). url: %s", url)
    return []schemas.JWKey{}
  }

  respBody, err3 := ies.requestSender.ReadAll(resp)
  if err3 != nil {
    log.Errorf("Error in IdentityExterService.GetJWKs. (err in readAll resp)")
    return []schemas.JWKey{}
  }

  jwkResp := &schemas.JWKResponse{}
  if err := Unpack(respBody, jwkResp); err != nil {
    log.Errorf("Error in IdentityExterService.GetJWKs. (err in unmarshalling resp)")
    return []schemas.JWKey{}
  }
  log.Infof("List len: %d", len(*jwkResp.Keys))

  return *jwkResp.Keys
}

func (ies IdentityExterService) RegisterUser(req *http.Request) (*schemas.AuthResp, *schemas.ErrorResponse) {
  url := ies.baseUrlApiv1 + config.SignUpHandler
  newReq, err := http.NewRequest(req.Method, url, req.Body)
  if err != nil {
    log.WithError(err).Errorf("Error in IdentityExterService.RegisterUser. (err in making request)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  resp, err2 := ies.requestSender.SendRequest(newReq)
  if err2 != nil {
    log.Errorf("Error in IdentityExterService.RegisterUser. (err in sending request)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
```

```
      Message:    "Internal Server error",
    }
  }

  respBody, err3 := ies.requestSender.ReadAll(resp)
  if err3 != nil {
    log.Errorf("Error in IdentityExterService.RegisterUser. (err in readAll resp)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  signUpResp := &schemas.AuthResp{}
  if err := Unpack(respBody, signUpResp); err != nil {
    errResp := &schemas.ErrorResponse{}
    if err := Unpack(respBody, errResp); err != nil {
      log.Errorf("Error in IdentityExterService.RegisterUser. (err in unmarshalling resp)")
      return nil, &schemas.ErrorResponse{
        StatusCode: 500,
        Message:    "Internal Server error",
      }
    }
    errResp.StatusCode = resp.StatusCode
    return nil, errResp
  }

  return signUpResp, nil
}

func (ies IdentityExterService) AuthUser(req *http.Request) (*schemas.AuthResp, *schemas.ErrorResponse) {
  url := ies.baseUrlApiv1 + config.AuthHandler
  newReq, err := http.NewRequest(req.Method, url, req.Body)
  if err != nil {
    log.WithError(err).Errorf("Error in IdentityExterService.AuthUser. (err in making request)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  resp, err2 := ies.requestSender.SendRequest(newReq)
  if err2 != nil {
    log.Errorf("Error in IdentityExterService.AuthUser. (err in sending request)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  respBody, err3 := ies.requestSender.ReadAll(resp)
  if err3 != nil {
    log.Errorf("Error in IdentityExterService.AuthUser. (err in readAll resp)")
    return nil, &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  authResp := &schemas.AuthResp{}
  if err := Unpack(respBody, authResp); err != nil {
    errResp := &schemas.ErrorResponse{}
    if err := Unpack(respBody, errResp); err != nil {
      log.Errorf("Error in IdentityExterService.AuthUser. (err in unmarshalling resp)")
      return nil, &schemas.ErrorResponse{
        StatusCode: 500,
        Message:    "Internal Server error",
      }
    }
    errResp.StatusCode = resp.StatusCode
    return nil, errResp
  }

  return authResp, nil
}

func (ies IdentityExterService) Logout(req *http.Request) *schemas.ErrorResponse {
  url := ies.baseUrlApiv1 + config.LogoutHandler
  newReq, err := http.NewRequest(req.Method, url, req.Body)
  newReq.Header = req.Header.Clone()
  if err != nil {
    log.WithError(err).Errorf("Error in IdentityExterService.Logout. (err in making request)")
```

```
    return &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  resp, err2 := ies.requestSender.SendRequest(newReq)
  if err2 != nil {
    log.Errorf("Error in IdentityExterService.Logout. (err in sending request)")
    return &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  respBody, err3 := ies.requestSender.ReadAll(resp)
  if err3 != nil {
    log.Errorf("Error in IdentityExterService.Logout. (err in readAll resp)")
    return &schemas.ErrorResponse{
      StatusCode: 500,
      Message:    "Internal Server error",
    }
  }

  if resp.StatusCode != http.StatusNoContent {
    errResp := &schemas.ErrorResponse{}
    if err := Unpack(respBody, errResp); err != nil {
      log.Errorf("Error in IdentityExterService.Logout. (err in unmarshalling resp)")
      return &schemas.ErrorResponse{
        StatusCode: 500,
        Message:    "Internal Server error",
      }
    }
    errResp.StatusCode = resp.StatusCode
    return errResp
  }
  return nil
}
```

Листинг 3.2: IdentityService

```
package externalServices

import (
  "container/list"
  "encoding/json"
  "fmt"
  "io"
  "net/http"
  "sync"
  "time"

  "bauman-poker/config"

  "github.com/go-playground/validator/v10"
  log "github.com/sirupsen/logrus"
)

type RequestSender struct {
  sendReq     func(req *http.Request) (*http.Response, error)
  mutex       *sync.Mutex
  context     *BreakerContext
  queue       *list.List
  isExecuting bool
}

func NewRequestSender(ctx *BreakerContext) *RequestSender {
  log.Info("RequestSender init. health-check url: ", ctx.urlH)
  return &RequestSender{
    sendReq: NewCircuitBreakerDecorator(ctx, http.DefaultClient.Do),
    context: ctx,
    mutex:   new(sync.Mutex),
    queue:   list.New(),
  }
}

func (sender RequestSender) SendRequest(req *http.Request) (*http.Response, error) {
  resp, err2 := sender.sendReq(req)
  if err2 != nil {
    if err2.Error() == "503" {
      log.Errorf("error in sending request. url: %s: %s\n", req.Method, req.URL.String())
```

```
    } else {
      log.Errorf("error response was got. url: %s: %s; status code = %d; status: %s\n", req.Method, req.URL.
     String(), resp.StatusCode, resp.Status)
    }
  } else {
    log.Infof("response received successful. url: %s: %s; status code = %d; status: %s\n", req.Method, req.
    URL.String(), resp.StatusCode, resp.Status)
  }
  return resp, err2
}

func Unpack(respBody []byte, respStructPtr any) error {
  if err4 := json.Unmarshal(respBody, respStructPtr); err4 != nil {
    log.WithError(err4).Errorf("error in unmarshalling response. ResponseBody: %#v\n", respBody)
    return fmt.Errorf("500")
  }

  if err := validator.New().Struct(respStructPtr); err != nil {
    log.WithError(err).Errorf("error in validate required fields in unpacking")
    return fmt.Errorf("500")
  }

  log.Infof("return response for request url")
  return nil
}

func (sender RequestSender) ReadAll(resp *http.Response) ([]byte, error) {
  respBody, err3 := io.ReadAll(resp.Body)
  if err3 != nil {
    log.WithError(err3).Errorf("error in reading response body. url: %s\n", resp.Request.URL.String())
    return nil, fmt.Errorf("500")
  }
  return respBody, nil
}

func (sender RequestSender) SendRequestForever(req *http.Request) error {
  resp, err2 := sender.sendReq(req)
  if err2 != nil {
    if err2.Error() == "503" {
      log.Errorf("error in sending request. url: %s: %s\n", req.Method, req.URL.String())
      sender.pushRequest(req)
      return nil
    } else {
      log.Errorf("error response was got. url: %s: %s; status code = %d; status: %s\n", req.Method, req.URL.
     String(), resp.StatusCode, resp.Status)
    }
  }
  return err2
}

func (sender RequestSender) pushRequest(request *http.Request) {
  sender.mutex.Lock()
  sender.queue.PushBack(request)
  log.Info(fmt.Sprintf("Request has added to queue. Requests in queue: %d", sender.queue.Len()))
  sender.mutex.Unlock()

  if !sender.isExecuting {
    sender.retryTimeout()
  }
}

func (sender RequestSender) retryTimeout() {
  sender.mutex.Lock()
  sender.isExecuting = true
  sender.mutex.Unlock()
  go func() {
    mutex := sender.mutex
    for !sender.isEmpty() {
      mutex.Lock()
      request := sender.queue.Front().Value.(*http.Request)
      mutex.Unlock()
      _, err := sender.sendReq(request)
      if err == nil || err.Error() != "503" {
        mutex.Lock()
        sender.queue.Remove(sender.queue.Front())
        log.Info(fmt.Sprintf("Request removed from queue. Requests in queue: %d", sender.queue.Len()))
        mutex.Unlock()
      } else {
        time.AfterFunc(config.Timeout*2, func() {
          sender.retryTimeout()
        })
        return
```

```
      }
    }

    if sender.isEmpty() {
      mutex.Lock()
      sender.isExecuting = false
      mutex.Unlock()
    }
  }()
}

func (sender RequestSender) isEmpty() bool {
  sender.mutex.Lock()
  len := sender.queue.Len()
  sender.mutex.Unlock()
  return len <= 0
}
```

Листинг 3.3: RequestSender

```
package handlers

import (
  pokergame "bauman-poker/poker-game"
  "bauman-poker/schemas"
  "bauman-poker/usecases"
  "bauman-poker/utils"
  "bytes"
  "encoding/json"
  "io"
  "strings"

  "net/http"

  "github.com/gin-gonic/gin"
  log "github.com/sirupsen/logrus"
)

const (
  HeaderAuth = "Authorization"
)

type GameServerHandler struct {
  uc        *usecases.GameServerUsecase
  validator *utils.TokenValidator
}

func NewGameServerHandler(uc *usecases.GameServerUsecase, v *utils.TokenValidator) *GameServerHandler {
  return &GameServerHandler{
    uc:        uc,
    validator: v,
  }
}

func (lp GameServerHandler) RegisterRoutes(router gin.IRouter) {
  router.POST("/register", lp.registerUser)
  router.POST("/oauth/token", lp.authUser)
  router.DELETE("/oauth/revoke", lp.logout)
  router.GET("/me", lp.validateToken, lp.getMe)
  router.GET("/rooms/matching", lp.validateToken, lp.matchingRoom)
  router.GET("/rooms/:roomUid", lp.validateToken, lp.getRoomInfo)
  router.GET("/players/:userUid", lp.validateToken, lp.getPlayerInfo)
  router.GET("/rooms-ws/:roomUid", lp.connectToRoom)
}

func (lp GameServerHandler) registerUser(context *gin.Context) {
  reqBytes, err := io.ReadAll(context.Request.Body)
  context.Request.Body.Close()
  if err != nil {
    log.WithError(err).Error("register Error in body readall")
    context.JSON(http.StatusBadRequest, schemas.ErrorResponse{
      Message: "Bad Request",
    })
    return
  }

  req := schemas.SignUpReq{}
  if err := json.Unmarshal(reqBytes, &req); err != nil {
    log.WithError(err).Error("register Error in Unmarshalling json")
    context.JSON(http.StatusBadRequest, schemas.ErrorResponse{
      Message: "Bad Request",
```

```
    })
    return
  }

  context.Request.Body = io.NopCloser(bytes.NewBuffer(reqBytes))
  resp, errResp := lp.uc.SignUpPlayer(context.Request, req.Username)

  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }

  context.JSON(http.StatusOK, resp)
}

func (lp GameServerHandler) authUser(context *gin.Context) {
  resp, errResp := lp.uc.AuthUser(context.Request)

  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }

  context.JSON(http.StatusOK, resp)
}

func (lp GameServerHandler) logout(context *gin.Context) {
  errResp := lp.uc.Logout(context.Request)
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  context.JSON(http.StatusNoContent, nil)
}

func (lp GameServerHandler) validateToken(context *gin.Context) {
  token := strings.Replace(context.GetHeader(HeaderAuth), "Bearer ", "", -1)
  if !lp.validator.VerifyAccessToken(token) {
    context.JSON(http.StatusUnauthorized, schemas.ErrorResponse{
      Message: "Unauthorized",
    })
    context.Abort()
    return
  }

  _, payload, _, _ := lp.validator.ParseAccessToken(token)
  context.Set("UserUid", payload.(*utils.AccessTokenPayload).UserUid)
  context.Next()
}

func (lp GameServerHandler) getMe(context *gin.Context) {
  userUid, _ := context.Get("UserUid")

  succResp, errResp := lp.uc.GetMe(userUid.(string))
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  context.JSON(http.StatusOK, succResp)
}

func (lp GameServerHandler) matchingRoom(context *gin.Context) {
  userUid, _ := context.Get("UserUid")

  succResp, errResp := lp.uc.MatchingRoom(userUid.(string))
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  context.JSON(http.StatusOK, succResp)
}

func (lp GameServerHandler) getRoomInfo(context *gin.Context) {
  userUid, _ := context.Get("UserUid")
  roomUid := context.Param("roomUid")

  succResp, errResp := lp.uc.GetRoomInfo(roomUid, userUid.(string))
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  context.JSON(http.StatusOK, succResp)
```

```
}

func (lp GameServerHandler) connectToRoom(context *gin.Context) {
  roomUid := context.Param("roomUid")
  userUid := context.Request.URL.Query().Get("uid")
  req := pokergame.WSRequest{
    RW:             context.Writer,
    Req:            context.Request,
    RoomUid:        roomUid,
    PlayerUid:      userUid,
    TokenValidator: lp.validator,
  }

  errResp := lp.uc.ConnectToRoom(req)
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  //context.Abort()
  //context.AbortWithStatus(http.StatusSwitchingProtocols)
}

func (lp GameServerHandler) getPlayerInfo(context *gin.Context) {
  playerUid := context.Param("userUid")
  pUidRelateOf, _ := context.Get("UserUid")
  succResp, errResp := lp.uc.GetPlayerInfo(playerUid, pUidRelateOf.(string))
  if errResp != nil {
    context.JSON(errResp.StatusCode, errResp)
    return
  }
  context.JSON(http.StatusOK, succResp)
}
```

Листинг 3.4: HTTP

```
package pokergame

import (
  "bauman-poker/repo"
  "container/list"
  "fmt"
  "time"

  log "github.com/sirupsen/logrus"
)

type GameBalancer struct {
  freeRooms          *list.List
  allRooms           map[string]*GameRoom // uid комнаты -> указатель на комнату
  playerToRoom       map[string]*GameRoom // распределение: uid игрока -> указатель на комнату
  unallocatedPlayers chan string
  repo               *repo.GormPlayerRepo
}

const (
  lenOfFreeRoomsQ       = 10
  lenOfAllocatedPlayerQ = 100
)

func NewGameBalancer(rp *repo.GormPlayerRepo) *GameBalancer {
  gb := &GameBalancer{
    freeRooms:          list.New(),
    allRooms:           make(map[string]*GameRoom),
    playerToRoom:       make(map[string]*GameRoom),
    unallocatedPlayers: make(chan string, lenOfAllocatedPlayerQ),
    repo:               rp,
  }

  go func() { // горутина распределения игроков по комнатам
    for {
      playerUid := <-gb.unallocatedPlayers
      if gb.playerToRoom[playerUid] != nil {
        continue
      }

      for {
        if gb.freeRooms.Len() <= 0 {
          room := newRoom(rp, gb)
          gb.freeRooms.PushBack(room)
          gb.allRooms[room.roomUid] = room
        }
```

```
        room := gb.freeRooms.Front().Value.(*GameRoom)

        if !room.addPlayer(playerUid) {
          gb.freeRooms.Remove(gb.freeRooms.Front())
        } else {
          gb.playerToRoom[playerUid] = room
          break
        }
      }
    }
  }()

  go func() { // горутина, удаляющая комнаты с завершенной игрой
    for {
      for u, r := range gb.allRooms {
        if r.roomState == DISSOLUTION {
          delete(gb.allRooms, u)
        }
      }
    }
  }()

  return gb
}

func (gb *GameBalancer) GetRoomUidByPlayer(playerUid string) string {
  if room := gb.playerToRoom[playerUid]; room != nil {
    return room.roomUid
  }
  return ""
}

func (gb *GameBalancer) MatchingRoom(playerUid string) *RoomInfo {
  if room := gb.playerToRoom[playerUid]; room != nil {
    return room.GetRoomInfo(playerUid)
  }
  gb.unallocatedPlayers <- playerUid

  room := gb.playerToRoom[playerUid]
  for room == nil {
    time.Sleep(10 * time.Millisecond)
    room = gb.playerToRoom[playerUid]
  }
  return room.GetRoomInfo(playerUid)
}

func (gb *GameBalancer) GetRoom(roomUid, playerUid string) *RoomInfo {
  return gb.allRooms[roomUid].GetRoomInfo(playerUid)
}

func (gb *GameBalancer) ConnectToRoom(wsReq WSRequest) bool {
  room := gb.allRooms[wsReq.RoomUid]

  if room == nil {
    log.Errorf("Error in GameBalancer.ConnectToRoom(). No such room with Uid: %s", wsReq.RoomUid)
    return false
  }

  return room.connectToRoom(wsReq)
}

func (gb *GameBalancer) GetPlayerInfo(playerUid, pUidRelateOf string) (*PlayerInfo, error) {
  if r := gb.playerToRoom[pUidRelateOf]; r != nil {
    if res := r.GetPlayerInfo(playerUid, pUidRelateOf); res != nil {
      return res, nil
    }
    return nil, fmt.Errorf("500")
  }
  return nil, fmt.Errorf("404")
}

func (gb *GameBalancer) removePlayerFromRoom(playerUid string) {
  delete(gb.playerToRoom, playerUid)
}
```

Листинг 3.5: GameBalancer

```
package pokergame

import "time"
```

```
type gameTimer struct {
  t          *time.Timer
  isAlarmed bool
}

func newGameTimer() *gameTimer {
  return &gameTimer{
    t:          nil,
    isAlarmed: false,
  }
}

/*
запускает таймер, если он не был запущен ранее. Сработает через <durationSec> секунд
*/
func (g *gameTimer) start(durationSec time.Duration) {
  if g.t != nil {
    return
  }

  g.reset()
  g.t = time.NewTimer(durationSec) // time.Duration(durationSec) * time.Second
  go func() {
    <-g.t.C
    g.isAlarmed = true
    g.t = nil
  }()
}

/*
останавливает таймер. Если он был остановлен ранее, то возвращается false, иначе -- true
*/
func (g *gameTimer) stop() bool {
  if g.t == nil {
    return false
  }
  f := g.t.Stop()
  g.t = nil
  return f
}

/*
устанавливает флаг срабатывания таймера в false
*/
func (g *gameTimer) reset() {
  g.isAlarmed = false
}
```

Листинг 3.6: GameTimer

```
package pokergame

import (
  "bauman-poker/repo"
  "container/list"
  "sync"

  log "github.com/sirupsen/logrus"
)

type Player struct {
  uid               string
  username          string
  bet               int64
  deposit           int64
  lastActionLabel   LastActionLabelType
  rank              repo.RankType
  personalCardList  *[]*PlayingCard
  boutVariants      *[]BoutVariantType
  bestComb          *BestComb
  //bestCombName      string
  timeEndBout    int64
  eventQ         *list.List
  lastEventIter *list.Element
  lastEvent      any
  lastEventMtx   *sync.Mutex
  wsConn         *WSConnection
  roomInfoCh     chan *RoomInfo   // канал, через который передается инфа о текущем состоянии комнаты
  playerInfoCh   chan *PlayerInfo // канал, через который передается инфа о запрошенном PlayerInfo
  isStartWanted bool
}
```

```
func newPlayer(userUid string, rp *repo.GormPlayerRepo, initEventQ *list.List) *Player {
  playerAcc, err := rp.GetPlayerByUid(userUid)
  if err != nil {
    log.Errorf("Error in Player.newPlayer()")
    return nil
  }
  if initEventQ == nil {
    log.Errorf("Error in Player.newPlayer(). initEvenQ is nil")
    return nil
  }
  return &Player{
    uid:               userUid,
    username:          playerAcc.Username,
    bet:               0,
    deposit:           StartDeposit,
    lastActionLabel:   LBL_NONE,
    rank:              playerAcc.UserRank,
    personalCardList:  &[]*PlayingCard{},
    boutVariants:      &[]BoutVariantType{},
    //bestCombName:      "",
    bestComb:        nil,
    timeEndBout:     0,
    eventQ:          initEventQ,
    lastEventIter:   nil, //initEventQ.Back(), // потому что все события до добавления игрока были переданы ему
      в *RoomInfo
    lastEvent:       nil,
    lastEventMtx:    new(sync.Mutex),
    roomInfoCh:      make(chan *RoomInfo),
    playerInfoCh:    make(chan *PlayerInfo),
    isStartWanted:   false,
  }
}

func (p *Player) pushEvent(e any) {
  // p.lastEvent =
  p.eventQ.PushBack(e)
  /*if p.wsConn != nil && !p.wsConn.isTerminated {
    p.wsConn.eventMsgManager(p.eventQ.PushBack(e))
  }*/
}

func (p *Player) setCurrentEvent(eventId int64) {
  id := int64(-100)
  for eIter := p.eventQ.Front(); eIter != nil; eIter = eIter.Next() {
    switch e := eIter.Value.(type) {
    case *GameEvent:
      id = int64(e.EventId)
    case *PrepareEvent:
      id = int64(e.EventId)
    case *PlayerActionEvent:
      id = int64(e.EventId)
    }
    if eventId == id {
      if eIter.Next() != nil {
        p.lastEventMtx.Lock()
        p.lastEventIter = eIter.Next()
        p.lastEvent = nil
        p.lastEventMtx.Unlock()
      } else {
        p.lastEventMtx.Lock()
        p.lastEventIter = eIter
        p.lastEvent = p.lastEventIter.Value
        p.lastEventMtx.Unlock()
      }
      break
    }
  }
}

/*
возвращает указатель на очередной Event на отправку. Если новых ивентов нет, то вернет nil
*/
func (p *Player) getNextEvent() any {
  defer p.lastEventMtx.Unlock()

  p.lastEventMtx.Lock()
  if p.lastEventIter == nil {
    if p.eventQ.Len() <= 0 {
      return nil
    }
    p.lastEventIter = p.eventQ.Front()
```

```
  }
  res := p.lastEventIter.Value
  if p.lastEventIter.Next() != nil {
    p.lastEventIter = p.lastEventIter.Next()
  }
  if p.lastEvent == res {
    return nil
  }

  p.lastEvent = res
  return res
}

func (p *Player) containsBoutVar(variantType BVariantType, coef CoefType) bool {
  for _, v := range *p.boutVariants {
    if v.VariantType == variantType {
      if v.VariantType == RAISE {
        for _, c := range *v.RaiseVariants {
          if c == coef {
            return true
          }
        }
        break
      } else {
        return true
      }
    }
  }
  return false
}

func (p *Player) resetForNewRound() {
  p.bet = 0
  p.lastActionLabel = LBL_NONE
  p.personalCardList = &[]*PlayingCard{}
  //p.bestCombName = ""
  p.bestComb = nil
}

func (p *Player) resetForNewTradeRound() {
  p.bet = 0
  if p.lastActionLabel != LBL_FOLD && p.lastActionLabel != LBL_ALLIN {
    p.lastActionLabel = LBL_NONE
  }
}

/*
Обновляет значение поля BoutVariants в зависимости от размера депозита, текущего размера ставки на столе.
     isCheck если true,

  будет добавлен вариант CHECK, иначе будет добавлен FOLD
*/
func (p *Player) updateBoutVars(currentBet int64) {
  if p.lastActionLabel == LBL_ALLIN || p.lastActionLabel == LBL_FOLD {
    return
  }

  bouts := list.New()
  if currentBet == p.bet {
    bouts.PushBack(BoutVariantType{
      VariantType: CHECK,
    })
  } else {
    bouts.PushBack(BoutVariantType{
      VariantType: FOLD,
    })
  }

  if currentBet-p.bet < p.deposit && currentBet != p.bet {
    bouts.PushBack(BoutVariantType{
      VariantType: CALL,
      CallValue:   currentBet - p.bet,
    })
  }

  if newBet := currentBet * 2; newBet-p.bet < p.deposit {
    bouts.PushBack(BoutVariantType{
      VariantType:    RAISE,
      RaiseVariants: &[]CoefType{X1_5, X2, ALLIN},
    })
  } else if newBet := 3 * currentBet / 2; newBet-p.bet < p.deposit {
    bouts.PushBack(BoutVariantType{
```

```
      VariantType:    RAISE,
      RaiseVariants: &[]CoefType{X1_5, ALLIN},
    })
  } else {
    bouts.PushBack(BoutVariantType{
      VariantType:    RAISE,
      RaiseVariants: &[]CoefType{ALLIN},
    })
  }

  newBouts := make([]BoutVariantType, bouts.Len())
  for bIter, idx := bouts.Front(), 0; bIter != nil; bIter, idx = bIter.Next(), idx+1 {
    newBouts[idx] = bIter.Value.(BoutVariantType)
  }
  p.boutVariants = &newBouts
}

func (p *Player) resetBoutVars() {
  p.boutVariants = &[]BoutVariantType{}
}

/*
Вносит блайнд, если возможно. Иначе делает ALL-IN. blindTypeAction тип блайнда:

  MIN_BLIND_IN | MAX_BLIND_IN
*/
func (p *Player) setBlind(blindSize int64, blindTypeAction ActType) *PlayerActionEvent {
  if blindSize < p.deposit {
    p.bet += blindSize
    p.deposit -= blindSize
    return &PlayerActionEvent{
      EventId:    GenId(),
      UserUid:    p.uid,
      ActionType: blindTypeAction,
      NewBet:     p.bet,
      NewDeposit: p.deposit,
    }
  } else {
    p.bet = p.deposit
    p.deposit = 0
    return &PlayerActionEvent{
      EventId:    GenId(),
      UserUid:    p.uid,
      ActionType: ALL_IN,
      NewBet:     p.bet,
      NewDeposit: p.deposit,
    }
  }
}

/*
На вход подается список открытых карт на столе. Функция ищет лучшую комбу среди открытых + своих карт
*/
func (p *Player) findBestComb(tableCards *[]*PlayingCard) {
  cards := make([]*PlayingCard, len(*tableCards)+len(*p.personalCardList))
  if len(cards) < 2 {
    return
  }
  log.Info("Try to find best Comb")

  copy(cards, *p.personalCardList)
  for i := range *tableCards {
    cards[i+2] = (*tableCards)[i]
  }

  p.bestComb = GetBestComb(cards)
  if p.bestComb != nil {
    log.Infof("BestCombName (p username: %s): %v", p.username, p.bestComb)
  } else {
    log.Infof("BestCombName (p username: %s): no comb :(", p.username)
  }
}
```

Листинг 3.7: Player

```
package pokergame

import "sort"

type SuitType string
```

```
const (
  DIAMONDS SuitType = "DIAMONDS"
  HEARTS   SuitType = "HEARTS"
  CLUBS    SuitType = "CLUBS"
  SPADES   SuitType = "SPADES"
)

type IndexType string

const (
  ACE   IndexType = "ACE"
  _2    IndexType = "2"
  _3    IndexType = "3"
  _4    IndexType = "4"
  _5    IndexType = "5"
  _6    IndexType = "6"
  _7    IndexType = "7"
  _8    IndexType = "8"
  _9    IndexType = "9"
  _10   IndexType = "10"
  JACK  IndexType = "JACK"
  QUEEN IndexType = "QUEEN"
  KING  IndexType = "KING"
)

type CombName string

/*const (
  ROYAL_FLUSH     string = "ROYAL-FLUSH"
  STRAIGHT_FLUSH  string = "STRAIGHT-FLUSH"
  KARE            string = "KARE"
  FULL_HOUSE      string = "FULL-HOUSE"
  FLUSH           string = "FLUSH"
  STRAIGHT        string = "STRAIGHT"
  TRIPLE          string = "TRIPLE"
  TWO_PAIR        string = "TWO PAIR"
  PAIR            string = "PAIR"
  HIGHEST_CARD    string = "HIGHEST CARD"
)*/

const (
  ROYAL_FLUSH     string = "РОЯЛ-ФЛЕШ"
  STRAIGHT_FLUSH  string = "СТРИТ-ФЛЕШ"
  KARE            string = "КАРЕ"
  FULL_HOUSE      string = "ФУЛЛ-ХАУС"
  FLUSH           string = "ФЛЕШ"
  STRAIGHT        string = "СТРИТ"
  TRIPLE          string = "ТРОЙКА"
  TWO_PAIR        string = "ДВЕ ПАРЫ"
  PAIR            string = "ПАРА"
  HIGHEST_CARD    string = "ВЫСШАЯ КАРТА"
)

type PlayingCard struct {
  CardSuit SuitType  `validate:"required"`
  Index    IndexType `validate:"required"`
  weight   int       `json:"-"`
  forCopy  bool      `json:"-"`
}

type BestComb struct {
  name   string
  cards  *[]*PlayingCard
  weight int
  wCard  int // вес старшей
}

var (
  cardSuits   = [...]SuitType{DIAMONDS, HEARTS, CLUBS, SPADES}
  cardIndexes = [...]IndexType{_2, _3, _4, _5, _6, _7, _8, _9, _10, JACK, QUEEN, KING, ACE}
  idxMap      = map[IndexType]int{_2: 2, _3: 3, _4: 4, _5: 5, _6: 6, _7: 7, _8: 8, _9: 9, _10: 10, JACK: 11,
    QUEEN: 12, KING: 13, ACE: 14}
)

func NewPlayingCard(suit SuitType, index IndexType) *PlayingCard {
  return &PlayingCard{
    CardSuit: suit,
    Index:    index,
    weight:   idxMap[index],
    forCopy:  false,
  }
}
```

```
func GetBestComb(inpCards []*PlayingCard) *BestComb {
  switch len(inpCards) {
  case 2:
    if res := isPair(inpCards); res != nil {
      return res
    }
    return isHighestCard(inpCards)
  case 5:
    return check5CardComb(inpCards)
  case 6:
    return check6CardComb(inpCards)
  case 7:
    return check7CardComb(inpCards)
  default:
    return nil
  }
}

func check7CardComb(inpCards []*PlayingCard) *BestComb {
  var maxComb *BestComb
  maxW := 0
  for idx := 0; idx < len(inpCards); idx++ {
    res := make([]*PlayingCard, 6)
    idx2 := 0
    for i := 0; i < len(inpCards); i++ {
      if idx != i {
        res[idx2] = inpCards[i]
        idx2++
      }
    }
    t := check6CardComb(res)
    if t != nil && t.weight > maxW {
      maxComb = t
      maxW = t.weight
    }
  }
  return maxComb
}

func check6CardComb(inpCards []*PlayingCard) *BestComb {
  var maxComb *BestComb
  maxW := 0
  for idx := 0; idx < len(inpCards); idx++ {
    res := make([]*PlayingCard, 5)
    idx2 := 0
    for i := 0; i < len(inpCards); i++ {
      if idx != i {
        res[idx2] = inpCards[i]
        idx2++
      }
    }
    t := check5CardComb(res)
    if t != nil && t.weight > maxW {
      maxComb = t
      maxW = t.weight
    }
  }
  return maxComb
}

func check5CardComb(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) != 5 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  sort.SliceStable(cards, func(a, b int) bool {
    return cards[a].weight < cards[b].weight
  })

  groups := make(map[IndexType][]*PlayingCard)
  for _, c := range cards {
    groups[c.Index] = append(groups[c.Index], c)
  }

  if res := isRoyalFlush(cards); res != nil {
    return res
  }
```

```
  switch len(groups) {
  case 2:
    for _, group := range groups {
      if len(group) == 4 {
        return &BestComb{
          name:   KARE,
          cards:  &group,
          weight: 8,
        }
      }
    }
    return &BestComb{
      name:   FULL_HOUSE,
      cards:  &cards,
      weight: 7,
    }
  case 3:
    for _, group := range groups {
      if len(group) == 3 {
        return &BestComb{
          name:   TRIPLE,
          cards:  &group,
          weight: 4,
        }
      }
    }
    return is2Pair(cards)
  case 4:
    return isPair(cards)
  default:
    flush := isFlush(cards)
    straight := isStraight(cards)
    switch {
    case flush && straight:
      return &BestComb{
        name:   STRAIGHT_FLUSH,
        cards:  &cards,
        weight: 9,
      }
    case flush:
      return &BestComb{
        name:   FLUSH,
        cards:  &cards,
        weight: 6,
      }
    case straight:
      return &BestComb{
        name:   STRAIGHT,
        cards:  &cards,
        weight: 5,
      }
    default:
      return isHighestCard(cards)
    }
  }

}

func isRoyalFlush(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) < 5 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  idxTypes := map[IndexType]int{ACE: 0, KING: 1, QUEEN: 2, JACK: 3, _10: 4}
  typesToIdx := map[int]IndexType{0: ACE, 1: KING, 2: QUEEN, 3: JACK, 4: _10}
  suitTypes := map[SuitType]int{DIAMONDS: 0, HEARTS: 1, CLUBS: 2, SPADES: 3}
  flags := [4][5]bool{}

  for idx := range cards {
    flags[suitTypes[cards[idx].CardSuit]][idxTypes[cards[idx].Index]] = true
  }

  for idx := 0; idx < len(flags); idx++ {
    isRes := true
    for j := 0; j < len(flags[idx]); j++ {
      if !flags[idx][j] {
        isRes = false
        break
      }
```

```
    }
    if isRes {
      res := make([]*PlayingCard, 5)
      for i := range flags[idx] {
        res[i] = NewPlayingCard(cardSuits[idx], typesToIdx[i])
      }
      return &BestComb{
        name:   ROYAL_FLUSH,
        cards:  &res,
        weight: 10,
        wCard:  14,
      }
    }
  }
  return nil
}

func isFlush(cards []*PlayingCard) bool {
  suit := cards[0].CardSuit
  for i := 1; i < 5; i++ {
    if cards[i].CardSuit != suit {
      return false
    }
  }
  return true
}

func isStraight(cards []*PlayingCard) bool {
  sorted := make([]*PlayingCard, 5)
  copy(sorted, cards)
  sort.Slice(sorted, func(i, j int) bool {
    return sorted[i].Index < sorted[j].Index
  })
  if sorted[0].weight+4 == sorted[4].weight {
    return true
  }
  if sorted[4].weight == 14 && sorted[0].weight == 2 && sorted[3].weight == 5 {
    return true
  }
  return false
}

/*func isStraignt(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) < 5 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  numOfMono := 0
  for idx := 0; idx < len(cards) - 1; idx++ {
    if cards[idx].weight + 1 == cards[idx + 1].weight {
      numOfMono++
    } else {
      numOfMono = 0
    }

  }
}*/

func isTriple(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) < 3 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  for idx := len(cards) - 1; idx > 1; {
    if cards[idx].Index == cards[idx-1].Index && cards[idx-1].Index == cards[idx-2].Index {
      cards[idx].forCopy = true
      cards[idx-1].forCopy = true
      cards[idx-2].forCopy = true
      idx -= 3
      break
    } else {
      idx--
    }
  }

  res := make([]*PlayingCard, 3)
```

```
  idx2 := 0
  maxW := 0
  for idx := len(cards) - 1; idx >= 0; idx-- {
    if cards[idx].forCopy {
      cards[idx].forCopy = false
      res[idx2] = cards[idx]
      if res[idx2].weight > maxW {
        maxW = res[idx2].weight
      }
      idx2++
    }
  }

  return &BestComb{
    name:   TRIPLE,
    cards:  &res,
    weight: 4,
    wCard:  maxW,
  }
}

func is2Pair(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) < 4 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  for idx := len(cards) - 1; idx > 0; {
    if cards[idx].Index == cards[idx-1].Index {
      cards[idx].forCopy = true
      cards[idx-1].forCopy = true
      idx -= 2
    } else {
      idx--
    }
  }

  res := make([]*PlayingCard, 4)
  idx2 := 0
  maxW := 0
  for idx := len(cards) - 1; idx >= 0; idx-- {
    if cards[idx].forCopy {
      cards[idx].forCopy = false
      res[idx2] = cards[idx]
      if res[idx2].weight > maxW {
        maxW = res[idx2].weight
      }
      idx2++
    }
  }

  return &BestComb{
    name:   TWO_PAIR,
    cards:  &res,
    weight: 3,
    wCard:  maxW,
  }
}

func isPair(inpCards []*PlayingCard) *BestComb {
  if len(inpCards) < 2 {
    return nil
  }

  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  for idx := len(cards) - 1; idx > 0; {
    if cards[idx].Index == cards[idx-1].Index {
      cards[idx].forCopy = true
      cards[idx-1].forCopy = true
      idx -= 2
      break
    } else {
      idx--
    }
  }

  res := make([]*PlayingCard, 2)
  idx2 := 0
```

```
  maxW := 0
  for idx := len(cards) - 1; idx >= 0; idx-- {
    if cards[idx].forCopy {
      cards[idx].forCopy = false
      res[idx2] = cards[idx]
      if res[idx2].weight > maxW {
        maxW = res[idx2].weight
      }
      idx2++
    }
  }

  return &BestComb{
    name:   PAIR,
    cards:  &res,
    weight: 2,
    wCard:  maxW,
  }
}

func isHighestCard(inpCards []*PlayingCard) *BestComb {
  cards := make([]*PlayingCard, len(inpCards))
  copy(cards, inpCards)

  sort.SliceStable(cards, func(a, b int) bool {
    return cards[a].weight < cards[b].weight
  })

  res := make([]*PlayingCard, 1)
  res[0] = cards[len(cards)-1]
  return &BestComb{
    name:   HIGHEST_CARD,
    cards:  &res,
    weight: 1,
    wCard:  res[0].weight,
  }
}
```

Листинг 3.8: PlayingCard

```
package pokergame

import (
  "bauman-poker/config"
  "bauman-poker/repo"
  "container/list"
  "math"
  "math/rand"
  "strings"
  "time"

  "github.com/google/uuid"
  log "github.com/sirupsen/logrus"
)

type RoomStateType string

const (
  FORMING      RoomStateType = "FORMING"
  GAMING       RoomStateType = "GAMING"
  DISSOLUTION  RoomStateType = "DISSOLUTION"
)

type GameRoom struct {
  gameBalancer       *GameBalancer
  roomUid            string
  roomState          RoomStateType
  playerMap          map[string]*list.Element //*Player
  playerList         *list.List
  deckOfCards        *[]*PlayingCard
  tableCardList      *[]*PlayingCard
  stack              int64
  boutIter           *list.Element //*Player
  dealerIter         *list.Element //*Player
  lastEventId        int64
  minBlind           int64
  maxBlind           int64
  roundNumber        int16
  tradeRoundNumber   int16
  msgQ               chan *ActionMessage
  numOfPlayers       int16
```

```
  numOfStartPlayers int16
  repo              *repo.GormPlayerRepo
  gTimer            *gameTimer
  cardIdx           int16

  ctrlBetSum int64 // контрольная сумма ставок.
  // В нее суммируются ставки игроков внутри круга торгов.
  // При BET_ACCEPTED сумма прибавляется к полю stack, а это поле становится 0

  currentBet     int64
  isIncreasedBet bool
  lastTraderIter *list.Element // указатель на игрока, на котором завершается текущий круг торгов
  //lastRaiserIter *list.Element // указатель на игрока, который последним поднимал ставку
}

func newRoom(repo *repo.GormPlayerRepo, gb *GameBalancer) *GameRoom {
  room := &GameRoom{
    gameBalancer:       gb,
    roomUid:            uuid.NewString(),
    roomState:          FORMING,
    playerMap:          make(map[string]*list.Element), //*Player
    playerList:         list.New(),
    deckOfCards:        &[]*PlayingCard{},
    tableCardList:      &[]*PlayingCard{},
    stack:              0,
    boutIter:           nil,
    dealerIter:         nil,
    lastEventId:        1,
    minBlind:           0, // StartMinBlind
    maxBlind:           0, // StartMinBlind * 2
    roundNumber:        0,
    tradeRoundNumber:   0,
    msgQ:               make(chan *ActionMessage, 1000),
    numOfPlayers:       0,
    numOfStartPlayers:  0,
    repo:               repo,
    gTimer:             newGameTimer(),
    cardIdx:            0,
    isIncreasedBet:     false,
  }

  go func() {
    for room.roomState != DISSOLUTION {
      switch room.roomState {
      case FORMING:
        room.forming()
      case GAMING:
        room.gaming()
      }
    }
  }()

  return room
}

func (gr *GameRoom) forming() {
  check := func() bool {
    return len(gr.playerMap) >= MaxNumOfPlayersPerGame ||
      (len(gr.playerMap) >= MinNumOfPlayersPerGame && (gr.numOfStartPlayers*2 > gr.numOfPlayers || gr.gTimer.
    isAlarmed))
  }

  for gr.roomState == FORMING {
    if check() {
      gr.setRoomState(GAMING)
      gr.gTimer.reset()
      return
    }
    if len(gr.msgQ) > 0 {
      msg := <-gr.msgQ
      if msg.MessageId == -1 { // запос на фиксацию состояния
        gr.fixRoomStateInfo(msg.UserUid)
        continue
      } else if msg.MessageId == -2 {
        u := strings.Split(msg.UserUid, ".")
        gr.fixPlayerInfo(u[0], u[1])
        continue
      }
      switch msg.MessageType {
      case VOTE:
        gr.voteActionHandling(msg)
      case GAME_ACTION:
```

```
        gr.ioPrepareActionHandling(msg)
      }
    }
  }
}

func (gr *GameRoom) gaming() {
  for gr.roundNumber = 1; gr.playerList.Len() > 1; gr.roundNumber++ { // цикл по раундам
    gr.notifyNewRound()
    gr.notifySetDealer()

    for gr.tradeRoundNumber = 1; gr.tradeRoundNumber <= 4 && gr.playerList.Len() > 1; gr.tradeRoundNumber++ {
      // цикл по торговым кругам
      gr.notifyNewTradeRound()
      if gr.tradeRoundNumber == 1 {
        gr.notifyMinBlindIn()
        gr.notifyMaxBlindIn()
        gr.notifyPersonalCards()
      } else {
        //gr.currentBet = gr.maxBlind
        //gr.lastTraderIter = gr.dealerIter
        gr.notifyTableCards()
        // выдача карт на стол воткрытую
      }
      for gr.playerList.Len() > 1 && !gr.isEndOfTradeRound() { // цикл торгового круга. Завершится, когда пос
    ледний из игроков, поднимавших ставку, выберет CHECK
        gr.notifyBout()
        p := gr.boutIter.Value.(*Player)
        gr.gTimer.start(time.Duration((p.timeEndBout - time.Now().UnixMilli()) * int64(time.Millisecond)))
        f := false
        for !f { // обрабатываем входящие сообщения, пока не получим действие игрока, у кого ход, или пока не
     истекло время таймера
          if gr.playerList.Len() <= 1 {
            break
          }
          select {
          case msg, ok := <-gr.msgQ:
            if ok {
              f = gr.playerActionHandling(msg)
              continue
            }
          default:
          }
          if gr.gTimer.isAlarmed {
            // делаем фолд или чек
            p := gr.boutIter.Value.(*Player)
            p.lastActionLabel = LBL_FOLD
            actType := ActType(FOLD)
            if p.containsBoutVar(CHECK, "ALL-IN") {
              actType = ActType(CHECK)
              p.lastActionLabel = LBL_CHECK
            }

            gr.broadcast(&PlayerActionEvent{
              EventId:    GenId(),
              UserUid:    p.uid,
              ActionType: actType,
            }, true)
            f = true
          }
        }
        gr.gTimer.stop()
        // принять ответ о действии игрока. Проверить, что это действие ему доступно.
        // Если CHECK, то выходим из цикла торгового круга
      }

      gr.notifyBetAccepted()
    }

    gr.notifyWinnerResult()
    log.Info("Winner-pause 10s.")
    time.Sleep(time.Second * WinnerResultPauseSecond)

    for _, pIter := range gr.playerMap {
      p := pIter.Value.(*Player)
      if p.deposit <= 0 {
        gr.outcomeUser(pIter)
      }
    }
  }
  gr.setRoomState(DISSOLUTION)
}
```

```
func (gr *GameRoom) isEndOfTradeRound() bool {
  if gr.boutIter == nil {
    return false
  }
  p := gr.boutIter.Value.(*Player)
  defer p.resetBoutVars()

  //---
  numOfFold := 0
  numOfALLIN := 0
  for _, pIter := range gr.playerMap {
    if pIter.Value.(*Player).lastActionLabel == LBL_FOLD {
      numOfFold++
    }
    if pIter.Value.(*Player).lastActionLabel == LBL_ALLIN {
      numOfALLIN++
    }
  }
  if numOfFold >= gr.playerList.Len()-1 || numOfALLIN >= gr.playerList.Len() {
    gr.tradeRoundNumber = 4
    gr.notifyTableCards()
    return true
  }
  //---

  if gr.boutIter == gr.lastTraderIter {
    /*if p.lastActionLabel == LBL_RAISE {
      return false
    }
    if p.lastActionLabel == LBL_ALLIN {
      return !gr.isIncreasedBet
      // он мог сделать ALL-IN:
      //  1) это повысило текущую ставку  -- продолжаем круг торгов (return false)
      //  2) не повысило текущую ставку -- принимаем ставки, идем на след. круг торгов (return true)
    }*/
    return !gr.isIncreasedBet
  }
  return false
}

func (gr *GameRoom) notifyNewRound() {
  for _, pIter := range gr.playerMap {
    pIter.Value.(*Player).resetForNewRound()
  }
  gr.tableCardList = &[]*PlayingCard{}

  gr.shuffleDeck()
  gr.recalcBlinds()
  gr.broadcast(&GameEvent{
    EventId:     GenId(),
    EventType:   NEW_ROUND,
    RoundNumber: gr.roundNumber,
  }, true)
}

func (gr *GameRoom) notifySetDealer() {
  gr.dealerIter = gr.nextPlayer(gr.dealerIter)
  gr.broadcast(&PlayerActionEvent{
    EventId:    GenId(),
    UserUid:    gr.dealerIter.Value.(*Player).uid,
    ActionType: SET_DEALER,
  }, true)
}

func (gr *GameRoom) notifyNewTradeRound() {
  gr.boutIter = nil
  gr.currentBet = 0
  gr.ctrlBetSum = 0
  gr.lastTraderIter = nil

  for _, pIter := range gr.playerMap {
    pIter.Value.(*Player).resetForNewTradeRound()
  }

  //gr.lastRaiserIter = nil
  gr.broadcast(&GameEvent{
    EventId:     GenId(),
    EventType:   NEW_TRADE_ROUND,
    RoundNumber: int16(gr.roundNumber),
  }, true)
}
```

```
func (gr *GameRoom) notifyMinBlindIn() {
  p := gr.nextPlayer(gr.dealerIter).Value.(*Player)

  delta := int64(math.Min(float64(gr.minBlind), float64(p.deposit)))
  gr.ctrlBetSum += delta
  gr.currentBet = int64(math.Max(float64(gr.currentBet), float64(gr.minBlind)))

  gr.broadcast(p.setBlind(gr.minBlind, MIN_BLIND_IN), true)
}

func (gr *GameRoom) notifyMaxBlindIn() {
  pIter := gr.nextPlayer(gr.nextPlayer(gr.dealerIter))
  p := pIter.Value.(*Player)

  delta := int64(math.Min(float64(gr.maxBlind), float64(p.deposit)))
  gr.ctrlBetSum += delta
  gr.currentBet = int64(math.Max(float64(gr.currentBet), float64(gr.maxBlind)))

  gr.broadcast(p.setBlind(gr.maxBlind, MAX_BLIND_IN), true)

  //gr.lastRaiserIter = pIter
  gr.lastTraderIter = pIter
}

func (gr *GameRoom) notifyPersonalCards() {
  eventId := GenId()
  for _, pIter := range gr.playerMap {
    p := pIter.Value.(*Player)
    cards := make([]*PlayingCard, 2)
    for idx := range cards {
      cards[idx] = gr.getCard()
    }
    p.personalCardList = &cards
    p.findBestComb(&[]*PlayingCard{})
    bestCombName := ""
    if p.bestComb != nil {
      bestCombName = p.bestComb.name
    }
    gr.sendPersonalEvent(&GameEvent{
      EventId:         eventId,
      EventType:       PERSONAL_CARDS,
      PlayingCardsList: p.personalCardList,
      BestCombName:    bestCombName,
    }, p, true)
  }
}

func (gr *GameRoom) notifyBout() {
  if gr.boutIter != nil {
    gr.boutIter = gr.nextPlayer(gr.boutIter)
  }
  for gr.boutIter == nil ||
    gr.boutIter.Value.(*Player).lastActionLabel == LBL_ALLIN ||
    gr.boutIter.Value.(*Player).lastActionLabel == LBL_FOLD {

    if gr.boutIter == nil {
      if gr.tradeRoundNumber == 1 {
        gr.boutIter = gr.nextPlayer(gr.nextPlayer(gr.nextPlayer(gr.dealerIter)))
      } else {
        gr.boutIter = gr.nextPlayer(gr.dealerIter)
        gr.lastTraderIter = gr.boutIter
      }
    } else {
      gr.boutIter = gr.nextPlayer(gr.boutIter)
    }
  }

  eventId := GenId()
  p := gr.boutIter.Value.(*Player)

  if gr.currentBet == 0 {
    p.updateBoutVars(gr.maxBlind)
  } else {
    p.updateBoutVars(gr.currentBet)
  }
  p.timeEndBout = time.Now().Add(time.Second * BoutTime).Add(time.Millisecond * time.Duration(config.
    MsgLeewayMilli)).UnixMilli()
  bestCombName := ""
  if p.bestComb != nil {
    bestCombName = p.bestComb.name
  }
```

```
  gr.sendPersonalEvent(&PlayerActionEvent{
    EventId:                eventId,
    UserUid:                p.uid,
    ActionType:             BOUT,
    BoutVariants:           p.boutVariants,
    BestCombName:           bestCombName,
    TimeEndBoutOrForming: p.timeEndBout,
  }, p, true)

  for _, anotherPlIter := range gr.playerMap {
    anotherP := anotherPlIter.Value.(*Player)
    if anotherP == p {
      continue
    }
    gr.sendPersonalEvent(&PlayerActionEvent{
      EventId:                eventId,
      UserUid:                p.uid,
      ActionType:             BOUT,
      TimeEndBoutOrForming: p.timeEndBout,
    }, anotherP, true)
  }
}

func (gr *GameRoom) notifyBetAccepted() {
  gr.stack += gr.ctrlBetSum
  gr.broadcast(&GameEvent{
    EventId:   GenId(),
    EventType: BET_ACCEPTED,
    NewStack:  gr.stack,
  }, true)
}

func (gr *GameRoom) notifyTableCards() {
  switch gr.tradeRoundNumber {
  case 2:
    deck := make([]*PlayingCard, 3)
    for idx := 0; idx < len(deck); idx++ {
      deck[idx] = gr.getCard()
    }
    gr.tableCardList = &deck
  case 3:
    deck := make([]*PlayingCard, 4)
    copy(deck, *gr.tableCardList)
    for idx := len(*gr.tableCardList); idx < len(deck); idx++ {
      deck[idx] = gr.getCard()
    }
    gr.tableCardList = &deck
  case 4:
    deck := make([]*PlayingCard, 5)
    copy(deck, *gr.tableCardList)
    for idx := len(*gr.tableCardList); idx < len(deck); idx++ {
      deck[idx] = gr.getCard()
    }
    gr.tableCardList = &deck
  default:
    return
  }

  eventId := GenId()
  for _, pIter := range gr.playerMap {
    p := pIter.Value.(*Player)
    bestCombName := ""
    p.findBestComb(gr.tableCardList)
    if p.bestComb != nil {
      bestCombName = p.bestComb.name
    }
    gr.sendPersonalEvent(&GameEvent{
      EventId:           eventId,
      EventType:         CARDS_ON_TABLE,
      PlayingCardsList: gr.tableCardList,
      BestCombName:      bestCombName,
    }, p, true)
  }
}

func (gr *GameRoom) notifyWinnerResult() {
  /*
    1) алгоритм определения наилучшей комбы и ее названия по списку карт (длины от 2, 5, 6, 7)
    2) алгоритм определения лучшей комбинации в списке нескольких комбинаций
    3) обновление в БД статистики игроков, покидающих комнату
  */
```

```
  idx := 0
  pUids := make([]string, gr.playerList.Len())
  closedCards := make([][]*PlayingCard, gr.playerList.Len())

  maxCombW := 0
  winners := list.New()
  for _, pIter := range gr.playerMap {
    p := pIter.Value.(*Player)
    pUids[idx] = p.uid
    closedCards[idx] = p.personalCardList
    idx++

    if p.bestComb != nil {
      if p.bestComb.weight > maxCombW {
        maxCombW = p.bestComb.weight
        winners = list.New()
        winners.PushBack(p)
      } else if p.bestComb.weight == maxCombW {
        winners.PushBack(p)
      }
    }
  }

  absWinners := list.New()
  maxWCard := 0
  for winIter := winners.Front(); winIter != nil; winIter = winIter.Next() {
    p := winIter.Value.(*Player)
    if p.bestComb != nil {
      if p.bestComb.wCard > maxWCard {
        maxWCard = p.bestComb.wCard
        absWinners = list.New()
        absWinners.PushBack(p)
      } else if p.bestComb.wCard == maxWCard {
        absWinners.PushBack(p)
      }
    }
  }

  winUids := make([]string, absWinners.Len())
  winDepos := make([]int64, absWinners.Len())
  bestCombs := make([][]*PlayingCard, absWinners.Len())
  bestCombName := ""
  idx = 0
  for winIter := absWinners.Front(); winIter != nil; winIter = winIter.Next() {
    p := winIter.Value.(*Player)
    p.deposit += gr.stack / int64(absWinners.Len())

    winUids[idx] = p.uid
    winDepos[idx] = p.deposit
    bestCombs[idx] = p.bestComb.cards
    bestCombName = p.bestComb.name
    idx++
  }
  gr.stack = 0

  gr.broadcast(&GameEvent{
    EventId:          GenId(),
    EventType:        WINNER_RESULT,
    ClosedCards:      &closedCards,
    PlayerUids:       &pUids,
    WinnerUids:       &winUids,
    BestCombinations: &bestCombs,
    BestCombName:     bestCombName,
    WinnerDeposits:   &winDepos,
    NewStack:         gr.stack,
  }, false)
}

func (gr *GameRoom) setRoomState(state RoomStateType) {
  log.Infof("Room %s: change state %s -> %s", gr.roomUid, gr.roomState, state)
  gr.roomState = state
  gr.broadcast(&GameEvent{
    EventId:      GenId(),
    EventType:    ROOM_STATE_UPDATE,
    NewRoomState: state,
  }, true)
}

func (gr *GameRoom) addPlayer(playerUid string) bool {
  if len(gr.playerMap) < MaxNumOfPlayersPerGame && gr.roomState == FORMING {
    gr.pushMsgQ(&ActionMessage{
      MessageType: GAME_ACTION,
```

```
      MessageId:    GenId(),
      RoomUid:      gr.roomUid,
      UserUid:      playerUid,
      ActionType:   INCOME,
    })
  }

  for len(gr.playerMap) < MaxNumOfPlayersPerGame && gr.roomState == FORMING {
    time.Sleep(10 * time.Millisecond)
    if gr.playerMap[playerUid] != nil {
      return true
    }
  }

  return false
}

func (gr *GameRoom) connectToRoom(wsReq WSRequest) bool {
  p := gr.playerMap[wsReq.PlayerUid].Value.(*Player)
  if p == nil {
    log.Errorf("Error in GameRoom.connectToRoom(). No such player with uid: %s", wsReq.PlayerUid)
    return false
  }
  if p.wsConn != nil && !p.wsConn.isTerminated {
    log.Errorf("Error in GameRoom.connectToRoom(). Old wsConn is not terminated")
    return false
  }

  p.wsConn = newWSConnection(wsReq.RW, wsReq.Req, wsReq.TokenValidator, p, gr)
  return p.wsConn != nil
}

/*
Добавляет сообщение *ActionMessage в очередь на обработку главной горутиной комнаты
*/
func (gr *GameRoom) pushMsgQ(msg *ActionMessage) bool {
  push := func(msg *ActionMessage) {
    gr.msgQ <- msg
  }

  check := []func() bool{
    func() bool { // запрос на фиксацию состояния комнаты или конкретного игрока
      return (msg.MessageId == -1 || msg.MessageId == -2) && msg.RoomUid == gr.roomUid
    },
    func() bool {
      return msg.RoomUid == gr.roomUid && msg.ActionType == INCOME && gr.roomState == FORMING
    },
    func() bool {
      return gr.validateMsg(msg)
    },
  }

  for _, ch := range check {
    if ch() {
      push(msg)
      return true
    }
  }

  return false
}

func (gr *GameRoom) fixRoomStateInfo(playerUid string) {
  p := gr.playerMap[playerUid].Value.(*Player)
  if p == nil {
    return
  }

  bout := ""
  if gr.boutIter != nil {
    bout = gr.boutIter.Value.(*Player).uid
  }

  dealer := ""
  if gr.dealerIter != nil {
    dealer = gr.dealerIter.Value.(*Player).uid
  }

  p.roomInfoCh <- &RoomInfo{
    RoomUid:          gr.roomUid,
    RoomState:        gr.roomState,
    PlayerList:       gr.makePlayerInfoList(p),
```

```
    TableCardList:      gr.tableCardList,
    Stack:              gr.stack,
    Bout:               bout,
    DealerUid:          dealer,
    LastEventId:        gr.lastEventId,
    NumOfPlayers:       gr.numOfPlayers,
    NumOfStartPlayers: gr.numOfStartPlayers,
  }
}

func (gr *GameRoom) fixPlayerInfo(pUid, pUidRelateOf string) {
  p := gr.playerMap[pUid].Value.(*Player)
  pRelateOf := gr.playerMap[pUidRelateOf].Value.(*Player)
  pRelateOf.playerInfoCh <- gr.playerToPlayerInfoRelateOf(p, pRelateOf)
}

/*
Формирует список инфомации об игроках. player *Player - игрок, относительно которого это делается
*/
func (gr *GameRoom) makePlayerInfoList(player *Player) *[]*PlayerInfo {
  res := make([]*PlayerInfo, gr.playerList.Len())
  for idx, iter := 0, gr.playerList.Front(); iter != nil; idx, iter = idx+1, iter.Next() {
    p := iter.Value.(*Player)

    res[idx] = gr.playerToPlayerInfoRelateOf(p, player)
  }

  return &res
}

/*
Выгружает PlayerInfo из Player. p -- указатель на игрока, инфо которого выгружаем. pRelateOf -- указатель на
    игрока для которого это делаем
*/
func (gr *GameRoom) playerToPlayerInfoRelateOf(p, pRelateOf *Player) *PlayerInfo {
  cardList := &[]*PlayingCard{}
  boutVars := &[]BoutVariantType{}
  bestCombName := ""
  timeEndBout := int64(0)
  voteType := WAIT

  if p.uid == pRelateOf.uid {
    cardList = p.personalCardList

    if p.bestComb != nil {
      bestCombName = p.bestComb.name
    }
  }

  switch e := p.eventQ.Back().Value.(type) {
  /*case *GameEvent:
  if e.EventType == WINNER_RESULT {
    cardList = p.personalCardList
  }*/
  case *PlayerActionEvent:
    if p.uid == pRelateOf.uid && e.ActionType == BOUT {
      boutVars = p.boutVariants
      timeEndBout = p.timeEndBout
    }
  }

  if p.isStartWanted {
    voteType = START
  }

  return &PlayerInfo{
    UserUid:           p.uid,
    Username:          p.username,
    ImageUrl:          "",
    Bet:               p.bet,
    Deposit:           p.deposit,
    LastActionLabel:   p.lastActionLabel,
    UserRank:          p.rank,
    PersonalCardList: cardList,
    BoutVariants:      boutVars,
    BestCombName:      bestCombName,
    TimeEndBout:       timeEndBout,
    VoteType:          voteType,
  }
}

/*
```

```
обработка сообщений-действий при roomState = GAMING. Возвращает true, если получено корректное сообщение от т
    ого игрока,
которому принадлежит очередь хода
*/
func (gr *GameRoom) playerActionHandling(msg *ActionMessage) bool {
  if gr.roomState != GAMING {
    return false
  }

  p := gr.boutIter.Value.(*Player)

  switch msg.MessageType {
  case GAME_ACTION:
    if msg.ActionType == OUTCOME {
      gr.outcomeUser(gr.playerMap[msg.UserUid])
      return p.uid == msg.UserUid
    } else {
      if p.uid != msg.UserUid {
        return false
      }
      if p.containsBoutVar(BVariantType(msg.ActionType), msg.Coef) {
        actType := ActType(msg.ActionType)
        gr.isIncreasedBet = false

        switch msg.ActionType {
        case PlayerActionType(FOLD):
          p.lastActionLabel = LBL_FOLD
        case PlayerActionType(CHECK):
          p.lastActionLabel = LBL_CHECK // завершать круг торгов
        case PlayerActionType(CALL):
          p.lastActionLabel = LBL_CALL

          if gr.currentBet == 0 {
            gr.isIncreasedBet = true
            gr.currentBet = gr.maxBlind
            gr.lastTraderIter = gr.boutIter
          }
          p.deposit -= (gr.currentBet - p.bet)
          gr.ctrlBetSum += (gr.currentBet - p.bet)
          p.bet = gr.currentBet
        case PlayerActionType(RAISE): // сдвигать сюда lastRaiserIter, lastTraderIter
          delta := int64(0)
          if msg.Coef == ALLIN {
            p.lastActionLabel = LBL_ALLIN
            actType = ALL_IN

            delta = p.deposit
            if delta+p.bet > gr.currentBet { // чекаем повышается ли ставка от ALL-IN игрока
              //gr.lastRaiserIter = gr.boutIter // сдвинул
              gr.lastTraderIter = gr.boutIter
              gr.isIncreasedBet = true
            }
          } else {
            p.lastActionLabel = LBL_RAISE
            if msg.Coef == X1_5 {
              delta = 3*gr.currentBet/2 - p.bet // p.bet
            } else if msg.Coef == X2 {
              delta = 2*gr.currentBet - p.bet // p.bet
            }

            //gr.lastRaiserIter = gr.boutIter // сдвинул
            gr.lastTraderIter = gr.boutIter
            gr.isIncreasedBet = true
          }

          p.bet += delta
          gr.ctrlBetSum += delta
          p.deposit -= delta
          gr.currentBet = int64(math.Max(float64(gr.currentBet), float64(p.bet)))
        }

        gr.broadcast(&PlayerActionEvent{
          EventId:    GenId(),
          UserUid:    p.uid,
          ActionType: actType,
          NewBet:     p.bet,
          NewDeposit: p.deposit,
        }, true)
        return true
      }
    }
  case VOTE:
```

```
    log.Info("VOTE is incorrect")
    return false
  }
  return false
}

/*
обработка сообщений входа/выхода при roomState = FORMING
*/
func (gr *GameRoom) ioPrepareActionHandling(msg *ActionMessage) {
  if gr.roomState != FORMING {
    return
  }

  switch msg.ActionType {
  case INCOME:
    p := newPlayer(msg.UserUid, gr.repo, gr.getInitEventQCopy())
    gr.playerMap[msg.UserUid] = gr.playerList.PushBack(p)
    gr.numOfPlayers++
    event := &PlayerActionEvent{
      EventId:    GenId(),
      UserUid:    msg.UserUid,
      ActionType: ActType(INCOME),
    }
    gr.broadcast(event, true)

    if len(gr.playerMap) >= MinNumOfPlayersPerGame {
      gr.gTimer.start(time.Duration(WaitStartTimeSecond * time.Second))
    }
  case OUTCOME:
    iter := gr.playerMap[msg.UserUid]
    if iter.Value.(*Player).isStartWanted {
      gr.numOfStartPlayers--
    }

    gr.outcomeUser(iter)
    if len(gr.playerMap) < MinNumOfPlayersPerGame {
      gr.gTimer.stop()
    }
  }
}

func (gr *GameRoom) getInitEventQCopy() *list.List {
  if len(gr.playerMap) <= 0 {
    return list.New()
  }
  res := list.New()
  keys := make([]string, 0, 1)
  for k := range gr.playerMap {
    keys = append(keys, k)
    break
  }
  p := gr.playerMap[keys[0]].Value.(*Player)
  res.PushBackList(p.eventQ)
  return res
}

/*
обработка сообщения-голосования при roomState = FORMING
*/
func (gr *GameRoom) voteActionHandling(msg *ActionMessage) {
  if gr.roomState != FORMING || msg.MessageType != VOTE {
    return
  }

  p := gr.playerMap[msg.UserUid].Value.(*Player)
  switch msg.VoteType {
  case START:
    if !p.isStartWanted {
      p.isStartWanted = true
      gr.numOfStartPlayers++
    }
  case WAIT:
    if p.isStartWanted {
      p.isStartWanted = false
      gr.numOfStartPlayers--
    }
  default:
    return
  }
  gr.broadcast(&PrepareEvent{
    EventId:            GenId(),
```

```
    NumOfPlayers:      gr.numOfPlayers,
    NumOfStartPlayers: gr.numOfStartPlayers,
  }, true)
}

func (gr *GameRoom) broadcast(event any, doUpdateLastEventId bool) {
  if doUpdateLastEventId {
    switch e := event.(type) {
    case *PlayerActionEvent:
      gr.lastEventId = e.EventId
    case *PrepareEvent:
      gr.lastEventId = e.EventId
    case *GameEvent:
      gr.lastEventId = e.EventId
    }
  }
  for _, p := range gr.playerMap {
    p.Value.(*Player).pushEvent(event)
  }
}

func (gr *GameRoom) sendPersonalEvent(event any, p *Player, doUpdateLastEventId bool) {
  if doUpdateLastEventId {
    switch e := event.(type) {
    case *PlayerActionEvent:
      gr.lastEventId = e.EventId
    case *PrepareEvent:
      gr.lastEventId = e.EventId
    case *GameEvent:
      gr.lastEventId = e.EventId
    }
  }
  p.pushEvent(event)
}

func GenId() int64 {
  time.Sleep(time.Nanosecond)
  return time.Now().UnixNano()
}

func (gr *GameRoom) GetRoomInfo(playerUid string) *RoomInfo {
  if p := gr.playerMap[playerUid].Value.(*Player); p != nil {
    gr.pushMsgQ(&ActionMessage{
      MessageId:   -1,
      MessageType: GAME_ACTION,
      RoomUid:     gr.roomUid,
      UserUid:     playerUid,
    })
    return <-p.roomInfoCh
  }

  return nil
}

/*
Получает PlayerInfo по playerUid. pUidRelateOf -- uid игрока, для которого делается выгрузка
*/
func (gr *GameRoom) GetPlayerInfo(playerUid, pUidRelateOf string) *PlayerInfo {
  if gr.playerMap[playerUid].Value.(*Player) == nil {
    return nil
  }
  if p := gr.playerMap[pUidRelateOf].Value.(*Player); p != nil {
    gr.pushMsgQ(&ActionMessage{
      MessageId:   -2,
      MessageType: GAME_ACTION,
      RoomUid:     gr.roomUid,
      UserUid:     playerUid + "." + pUidRelateOf,
    })
    return <-p.playerInfoCh
  }
  return nil
}

func (gr *GameRoom) validateMsg(msg *ActionMessage) bool {
  if msg.RoomUid != gr.roomUid {
    return false
  }
  iterP := gr.playerMap[msg.UserUid]
  if iterP == nil {
    return false
  }
```

```
  p := iterP.Value.(*Player)
  switch msg.MessageType {
  case VOTE:
    return gr.roomState == FORMING
  case GAME_ACTION:
    if gr.roomState == DISSOLUTION {
      return false
    }
    if msg.ActionType == PlayerActionType(CHECK) || msg.ActionType == PlayerActionType(FOLD) ||
      msg.ActionType == PlayerActionType(CALL) || msg.ActionType == PlayerActionType(RAISE) {
      if gr.boutIter.Value.(*Player).uid != p.uid {
        return false
      }
      return p.containsBoutVar(BVariantType(msg.ActionType), msg.Coef)
    }
  default:
    return false
  }
  return true
}

/*
Возвращает указатель на следующего игрока относительно переданного в параметре
*/
func (gr *GameRoom) nextPlayer(playerIter *list.Element) *list.Element {
  if playerIter == nil {
    return gr.playerList.Front()
  } else {
    res := playerIter.Next()
    if res == nil {
      res = gr.playerList.Front()
    }
    return res
  }
}

/*
пересчитывает размеры блайндов каждые 2 раунда
*/
func (gr *GameRoom) recalcBlinds() {
  gr.minBlind = int64(StartMinBlind) * (int64(gr.roundNumber)/2 + 1)
  gr.maxBlind = 2 * gr.minBlind
}

/*
инициализирует колоду
*/
func (gr *GameRoom) initCardDeck() {
  deck := make([]*PlayingCard, len(cardSuits)*len(cardIndexes))
  for idxS, suit := range cardSuits {
    for idxI, index := range cardIndexes {
      deck[idxS*len(cardIndexes)+idxI] = NewPlayingCard(suit, index)
    }
  }
  gr.deckOfCards = &deck
}

/*
тассует колоду
*/
func (gr *GameRoom) shuffleDeck() {
  gr.initCardDeck()
  gen := rand.New(rand.NewSource(time.Now().UnixNano()))
  rand := func() uint64 {
    return gen.Uint64()
  }

  for i := len(*gr.deckOfCards) - 1; i > 1; i-- {
    j := rand() % uint64(i+1)
    (*gr.deckOfCards)[i], (*gr.deckOfCards)[j] = (*gr.deckOfCards)[j], (*gr.deckOfCards)[i]
  }
  gr.cardIdx = 0
}

/*
Возвращает указатель на верхнюю карту колоды, "изымая" карту из колоды. И переходит к следующей карте
*/
func (gr *GameRoom) getCard() *PlayingCard {
  if gr.cardIdx >= int16(len(*gr.deckOfCards)) {
    return nil
  }
  gr.cardIdx++
```

```
  return (*gr.deckOfCards)[gr.cardIdx-1]
}

/*
обрабатывает событие покидания комнаты игроком
*/
func (gr *GameRoom) outcomeUser(playerIterator *list.Element) {
  p := playerIterator.Value.(*Player)

  gr.gameBalancer.removePlayerFromRoom(p.uid)

  if gr.roomState == GAMING {
    if playerIterator == gr.dealerIter {
      if playerIterator.Prev() != nil {
        gr.dealerIter = playerIterator.Prev()
      } else {
        gr.dealerIter = gr.playerList.Back()
      }
    }
    if playerIterator == gr.boutIter {
      if playerIterator.Next() != nil {
        gr.boutIter = playerIterator.Next()
      } else {
        gr.boutIter = gr.playerList.Back()
      }
    }
    if playerIterator == gr.lastTraderIter {
      if playerIterator.Prev() != nil {
        gr.lastTraderIter = playerIterator.Prev()
      } else {
        gr.lastTraderIter = gr.playerList.Back()
      }
    }
    /*if playerIterator == gr.lastRaiserIter {
      gr.lastRaiserIter = nil
    }*/
  }

  gr.playerList.Remove(playerIterator)
  delete(gr.playerMap, p.uid)
  gr.numOfPlayers--
  event := &PlayerActionEvent{
    EventId:    GenId(),
    UserUid:    p.uid,
    ActionType: ActType(OUTCOME),
  }
  //p.wsConn.close()
  gr.broadcast(event, true)
}
```

Листинг 3.9: Room

```
package pokergame

import (
  "bauman-poker/repo"
  "bauman-poker/utils"
  "net/http"
)

type WSRequest struct {
  RW             http.ResponseWriter
  Req            *http.Request
  RoomUid        string
  PlayerUid      string
  TokenValidator *utils.TokenValidator
}

type LastActionLabelType string

const (
  LBL_NONE  LastActionLabelType = "NONE"
  LBL_FOLD  LastActionLabelType = "FOLD"
  LBL_CHECK LastActionLabelType = "CHECK"
  LBL_CALL  LastActionLabelType = "CALL"
  LBL_RAISE LastActionLabelType = "RAISE"
  LBL_ALLIN LastActionLabelType = "ALL-IN"
)

type PlayerInfo struct {
  UserUid          string `validate:"required"`
```

```
  Username          string `validate:"required"`
  ImageUrl          string
  Bet               int64              `validate:"required"`
  Deposit           int64              `validate:"required"`
  LastActionLabel   LastActionLabelType `validate:"required"`
  UserRank          repo.RankType      `validate:"required"`
  PersonalCardList  *[]*PlayingCard    `validate:"required"`
  BestCombName      string              `validate:"required"`
  BoutVariants      *[]BoutVariantType  `validate:"required"`
  TimeEndBout       int64               `validate:"required"`
  VoteType          PlayerVoteType      `validate:"required"`
}

type RoomInfo struct { // состояние комнаты с точки зрения одного из игроков
  RoomUid           string          `validate:"required"`
  RoomState         RoomStateType   `validate:"required"`
  PlayerList        *[]*PlayerInfo  `validate:"required"`
  TableCardList     *[]*PlayingCard `validate:"required"`
  Stack             int64           `validate:"required"`
  Bout              string          `validate:"required"`
  DealerUid         string          `validate:"required"`
  LastEventId       int64           `validate:"required"`
  RoundNumber       int64           `validate:"required"`
  NumOfPlayers      int16           `validate:"required"`
  NumOfStartPlayers int16           `validate:"required"`
}
```

Листинг 3.10: Schemas

```
package pokergame

import (
  "bauman-poker/config"
  external "bauman-poker/external-services"
  "bauman-poker/utils"
  "container/list"
  "net/http"
  "time"

  "github.com/gorilla/websocket"
  log "github.com/sirupsen/logrus"
)

type WSConnection struct {
  ws              *websocket.Conn
  player          *Player
  room            *GameRoom
  tokenValidator *utils.TokenValidator
  expAuthTime     int64
  msgQForSend     list.List //chan any
  respMsgQ        chan *ResponseMessage
  isSending       bool
  isTerminated    bool
  lastPingTime    int64 // момент времени последнего пинга в мс
  lastMsgTime     int64 // момент времени получения последнего сообщения от клиента
}

var upgrader = websocket.Upgrader{
  ReadBufferSize:  1024,
  WriteBufferSize: 1024,
}

func newWSConnection(rw http.ResponseWriter, r *http.Request, tv *utils.TokenValidator, p *Player, gr *
    GameRoom) *WSConnection {
  ws, err := upgrader.Upgrade(rw, r, nil)
  if err != nil {
    log.WithError(err).Error("error in creating WS-Connection")
    return nil
  }

  wsc := &WSConnection{
    ws:              ws,
    player:          p,
    room:            gr,
    tokenValidator: tv,
    expAuthTime:     0,
    msgQForSend:     *list.New(), //make(chan any, 1000),
    respMsgQ:        make(chan *ResponseMessage),
    isSending:       false,
    isTerminated:    false,
    lastPingTime:    0,
```

```
    lastMsgTime:     time.Now().UnixMilli(),
  }
  wsc.wsReader()
  wsc.wsMsgSender()

  return wsc
}

func (wsc *WSConnection) wsReader() {
  go func() {
    defer wsc.close()

    for !wsc.isTerminated {
      log.Info("next read")
      _, bmsg, err := wsc.ws.ReadMessage()
      wsc.lastMsgTime = time.Now().UnixMilli()
      log.Infof("%v", bmsg)

      if err != nil {
        log.WithError(err).Errorf("Error in reading msg")
        return
      } /*else if msgType != websocket.TextMessage {
        log.Errorf("Expected type: Text message!")
        return
      } */

      switch msg := UnpackMsgFromPlayer(bmsg).(type) {
      case (*PongMessage):
        log.Info("Casted to PongMsg")
      case (*AuthMessage):
        log.Info("Casted to AuthMsg")
        if !wsc.updateAuthToken(*msg) {
          log.Error("Auth-error")
        }
      case (*ActionMessage):
        log.Info("Casted to ActionMsg")
        if !wsc.authIsExpired() {
          if msg.UserUid != wsc.player.uid || !wsc.room.pushMsgQ(msg) {
            wsc.sendRespMsg(msg.MessageId, StatusBadReq)
          } else {
            wsc.sendRespMsg(msg.MessageId, StatusOK)
          }
        } else {
          wsc.sendRespMsg(msg.MessageId, StatusUnauthorized)
        }
      default:
        log.Info("Untyped msg")
      }
    }
  }()

  go func() {
    for !wsc.isTerminated {
      if time.Now().UnixMilli()-wsc.lastMsgTime > config.PingPeriodMilli+config.MsgLeewayMilli {
        log.Errorf("PONG TIME ERROR")
        wsc.close()
        return
      }
    }
  }()
}

func NewEventMessage(event any) any {
  switch e := event.(type) {
  case *GameEvent:
    return &EventMessage[GameEvent]{
      MessageType:     EVENT,
      MessageId:       GenId(),
      EventType:       GAME_EVENT,
      EventDescriptor: e,
    }
  case *PrepareEvent:
    return &EventMessage[PrepareEvent]{
      MessageType:     EVENT,
      MessageId:       GenId(),
      EventType:       PREPARE_EVENT,
      EventDescriptor: e,
    }
  case *PlayerActionEvent:
    return &EventMessage[PlayerActionEvent]{
      MessageType:     EVENT,
      MessageId:       GenId(),
```

```
      EventType:       PLAYER_ACTION_EVENT,
      EventDescriptor: e,
    }
  }

  return nil
}

func (wsc *WSConnection) authIsExpired() bool {
  return wsc.expAuthTime+config.LeewaySeconds < time.Now().Unix()
}

func (wsc *WSConnection) updateAuthToken(msg AuthMessage) bool {
  if !wsc.tokenValidator.VerifyAccessToken(msg.Token) {
    wsc.sendRespMsg(msg.MessageId, StatusUnauthorized)
    return false
  }
  _, p, _, _ := wsc.tokenValidator.ParseAccessToken(msg.Token)
  payload := p.(*utils.AccessTokenPayload)
  if payload.UserUid != wsc.player.uid {
    wsc.sendRespMsg(msg.MessageId, StatusUnauthorized)
    return false
  }

  wsc.sendRespMsg(msg.MessageId, StatusOK)

  //if wsc.authIsExpired() {
  wsc.player.setCurrentEvent(msg.LastEventId)
  //}
  wsc.expAuthTime = payload.Exp
  return true
}

func (wsc *WSConnection) wsMsgSender() {
  wsc.isSending = true

  send := func(msg any) {
    if wsc.ws == nil {
      wsc.isSending = false
      wsc.isTerminated = true
    }
    if err := wsc.ws.WriteJSON(msg); err != nil {
      log.WithError(err).Errorf("Error in wsMsgSender")
      wsc.close()
      /*if wsc.ws != nil {
        wsc.ws.Close()
      }
      wsc.isSending = false
      wsc.isTerminated = true*/
      return
    }
  }

  go func() {
    for !wsc.isTerminated {
      if time.Now().UnixMilli()-wsc.lastPingTime > config.PingPeriodMilli {
        wsc.lastPingTime = time.Now().UnixMilli()
        send(&PingMessage{
          MessageType: PING,
        })
      }

      select {
      case msg, ok := <-wsc.respMsgQ:
        if ok {
          log.Info("try to send respMsg")
          send(msg)
        }
      default:
      }

      if !wsc.authIsExpired() {
        if event := wsc.player.getNextEvent(); event != nil {
          msg := NewEventMessage(event)
          send(msg)
          if wsc.isSelftCloseEvent(event) {
            wsc.close()
            return
          }
        }
      }
    }
```

```
  }()
}

func (wsc *WSConnection) sendRespMsg(msgId int64, statusCode RespStatusCodeType) {
  wsc.respMsgQ <- &ResponseMessage{
    MessageType:  ACK,
    AckMessageId: msgId,
    StatusCode:   statusCode,
  }
}

func UnpackMsgFromPlayer(bmsg []byte) any {
  pongMsg := &PongMessage{}
  authMsg := &AuthMessage{}
  actionMsg := &ActionMessage{}
  if err2 := external.Unpack(bmsg, authMsg); err2 == nil {
    return authMsg
  } else if err3 := external.Unpack(bmsg, actionMsg); err3 == nil {
    return actionMsg
  } else if err := external.Unpack(bmsg, pongMsg); err == nil {
    return pongMsg
  } else {
    log.WithError(err).WithError(err2).WithError(err3).Errorf("Errors in unpacking msgs from player")
    return nil
  }
}

func (wsc *WSConnection) isSelftCloseEvent(event any) bool {
  switch e := event.(type) {
  case *PlayerActionEvent:
    if e.ActionType == ActType(OUTCOME) {
      return e.UserUid == wsc.player.uid
    }
  }
  return false
}

func (wsc *WSConnection) close() {
  wsc.isTerminated = true
  wsc.isSending = false
  if wsc.ws != nil {
    wsc.ws.Close()
  }
}
```

Листинг 3.11: WS

```
package pokergame

type PongMessageType string

const PONG PongMessageType = "PONG"

type PongMessage struct {
  MessageType PongMessageType `validate:"required"`
  //MessageId   int64          `validate:"required"`
}

type AuthMessageType string

const AUTH AuthMessageType = "AUTH"

type AuthMessage struct {
  MessageType AuthMessageType `validate:"required"`
  MessageId   int64           `validate:"required"`
  RoomUid     string          `validate:"required"`
  Token       string          `validate:"required"`
  LastEventId int64           `validate:"required"`
}

type ActionMsgType string

const (
  GAME_ACTION ActionMsgType = "GAME-ACTION"
  VOTE        ActionMsgType = "VOTE"
)

type PlayerActionType BVariantType

const (
  INCOME  PlayerActionType = "INCOME"
```

```
  OUTCOME PlayerActionType = "OUTCOME"
)

type CoefType string

const (
  X1_5  CoefType = "X1_5"
  X2    CoefType = "X2"
  ALLIN CoefType = "ALL-IN"
)

type PlayerVoteType string

const (
  START PlayerVoteType = "START"
  WAIT  PlayerVoteType = "WAIT"
)

type ActionMessage struct {
  MessageType ActionMsgType `validate:"required"`
  MessageId   int64         `validate:"required"`
  RoomUid     string        `validate:"required"`
  UserUid     string        `validate:"required"`
  ActionType  PlayerActionType
  Coef        CoefType
  VoteType    PlayerVoteType
}
```

Листинг 3.12: WS Player

```
package pokergame

type PingMessageType string

const PING PingMessageType = "PING"

type PingMessage struct {
  MessageType PingMessageType `validate:"required"`
  //MessageId   int64           `validate:"required"`
}

type ResponseMsgType string

const ACK ResponseMsgType = "ACK"

type RespStatusCodeType int16

const (
  StatusOK           RespStatusCodeType = 200
  StatusUnauthorized RespStatusCodeType = 401
  StatusBadReq       RespStatusCodeType = 400
)

type ResponseMessage struct {
  MessageType  ResponseMsgType    `validate:"required"`
  AckMessageId int64              `validate:"required"`
  StatusCode   RespStatusCodeType `validate:"required"`
}

type EventMsgType string

const EVENT EventMsgType = "EVENT"

type BVariantType string

const (
  FOLD  BVariantType = "FOLD"
  CHECK BVariantType = "CHECK"
  CALL  BVariantType = "CALL"
  RAISE BVariantType = "RAISE"
)

type ActType PlayerActionType

const (
  //INCOME ActType = "INCOME"
  BOUT          ActType = "BOUT"
  ALL_IN        ActType = "ALL-IN"
  SET_DEALER    ActType = "SET-DEALER"
  MIN_BLIND_IN  ActType = "MIN-BLIND-IN"
  MAX_BLIND_IN  ActType = "MAX-BLIND-IN"
```

```
)

type BoutVariantType struct {
  VariantType    BVariantType `validate:"required"`
  CallValue      int64
  RaiseVariants  *[]CoefType
}

type PlayerActionEvent struct {
  EventId               int64    `validate:"required"`
  UserUid               string   `validate:"required"`
  ActionType            ActType  `validate:"required"`
  BoutVariants          *[]BoutVariantType
  BestCombName          string
  TimeEndBoutOrForming  int64
  NewBet                int64
  NewDeposit            int64
}

type PrepareEvent struct {
  EventId           int64  `validate:"required"`
  NumOfPlayers      int16  `validate:"required"`
  NumOfStartPlayers int16  `validate:"required"`
}

type GameEventType string

const (
  ROOM_STATE_UPDATE GameEventType = "ROOM_STATE_UPDATE"
  NEW_ROUND         GameEventType = "NEW_ROUND"
  NEW_TRADE_ROUND   GameEventType = "NEW_TRADE_ROUND"
  PERSONAL_CARDS    GameEventType = "PERSONAL_CARDS"
  CARDS_ON_TABLE    GameEventType = "CARDS_ON_TABLE"
  BET_ACCEPTED      GameEventType = "BET_ACCEPTED"
  WINNER_RESULT     GameEventType = "WINNER_RESULT"
)

type GameEvent struct {
  EventId           int64          `validate:"required"`
  EventType         GameEventType  `validate:"required"`
  NewRoomState      RoomStateType
  RoundNumber       int16
  PlayingCardsList  *[]*PlayingCard
  ClosedCards       *[]*[]*PlayingCard
  PlayerUids        *[]string
  WinnerUids        *[]string
  BestCombinations  *[]*[]*PlayingCard
  BestCombName      string
  WinnerDeposits    *[]int64
  NewStack          int64
}

type TypeOfEvent string

const (
  PREPARE_EVENT       TypeOfEvent = "PREPARE-EVENT"
  GAME_EVENT          TypeOfEvent = "GAME-EVENT"
  PLAYER_ACTION_EVENT TypeOfEvent = "PLAYER-ACTION-EVENT"
)

type EventMessage[T PlayerActionEvent | PrepareEvent | GameEvent] struct {
  MessageType     EventMsgType  `validate:"required"`
  MessageId       int64         `validate:"required"`
  EventType       TypeOfEvent   `validate:"required"`
  EventDescriptor *T            `validate:"required"`
}
```

Листинг 3.13: WS Server

```
package utils

type TokenType string

const (
  ACCESS TokenType = "ACCESS"
)

type Header struct {
  Alg       string    `json:"alg"`
  Typ       string    `json:"typ"`
  Kid       string    `json:"kid"`
```

```
  TokenType TokenType `json:"token_type"`
}

type AccessTokenPayload struct {
  Jti      string `json:"jti"`
  UserUid  string `json:"user_uid"`
  Iss      string `json:"iss"`
  Iat      int64  `json:"iat"`
  Exp      int64  `json:"exp"`
  DeviceId string `json:"device_id"`
}
```

Листинг 3.14: Token

```
package utils

import (
  "bauman-poker/config"
  externalServices "bauman-poker/external-services"
  "bauman-poker/schemas"
  "crypto/rsa"
  "encoding/base64"
  "encoding/binary"
  "encoding/json"
  "fmt"
  "math/big"
  "strings"
  "time"

  "github.com/dgrijalva/jwt-go"
  log "github.com/sirupsen/logrus"
)

type TokenValidator struct {
  jwks             []schemas.JWKey
  identityProvider *externalServices.IdentityExterService
}

func NewTokenValidator(identityProv *externalServices.IdentityExterService) *TokenValidator {
  return &TokenValidator{
    jwks:             []schemas.JWKey{},
    identityProvider: identityProv,
  }
}

func (tv *TokenValidator) getJWKs() {
  if len(tv.jwks) <= 0 {
    tv.jwks = tv.identityProvider.GetJWKs()
    log.Infof("List len: %d", len(tv.jwks))
    log.Infof("key[0]: %v", tv.jwks[0])
  }
}

func (tv *TokenValidator) getJWK(kid string) *schemas.JWKey {
  tv.getJWKs()
  key := tv.getKeyById(kid)

  if key == nil {
    log.Infof("Key is nil")
    tv.jwks = []schemas.JWKey{}
    tv.getJWKs()
    key := tv.getKeyById(kid)
    if key == nil {
      return nil
    }
  }
  return key
}

func (tv *TokenValidator) getKeyById(kid string) *schemas.JWKey {
  for _, key := range tv.jwks {
    log.Infof("Kid: %s", key.Kid)
    if key.Kid == kid {
      return &key
    }
  }
  return nil
}

func (tv *TokenValidator) VerifyAccessToken(token string) bool {
  header, payload, _, err := tv.ParseAccessToken(token)
```

```
  if err != nil {
    log.Errorf("Error in repo.verifyAccessToken")
    return false
  }
  key := tv.getJWK(header.(*Header).Kid)
  if key == nil {
    log.Error("Error in getting PubKey. repo.verifyAccessToken")
    return false
  }

  pubKey := jwkeyToPubKey(key)
  if pubKey == nil {
    return false
  }

  f1 := tv.verifyAlg(header.(*Header))
  f2 := tv.verifyTyp(header.(*Header))
  f3 := tv.verifyTokenType(header.(*Header))
  f4 := tv.verifyIss(payload.(*AccessTokenPayload))
  f5 := tv.verifyExp(payload.(*AccessTokenPayload))
  f6 := tv.verifySignature(token, pubKey, header.(*Header).Alg)
  log.Infof("verification flags: %t, %t, %t, %t, %t, %t", f1, f2, f3, f4, f5, f6)

  if !f1 || !f2 || !f3 || !f4 || !f5 || !f6 {
    return false
  }

  return true
}

func jwkeyToPubKey(key *schemas.JWKey) *rsa.PublicKey {
  e_bytes, err := base64.URLEncoding.DecodeString(key.E)
  if err != nil {
    log.WithError(err).Errorf("Error in casting utils.JWKeyToPubKey. (E)")
    return nil
  }
  e := binary.LittleEndian.Uint32(e_bytes)

  n_bytes, err2 := base64.URLEncoding.DecodeString(key.N)
  if err2 != nil {
    log.WithError(err2).Errorf("Error in casting utils.JWKeyToPubKey. (N)")
    return nil
  }
  n := new(big.Int)
  n.SetBytes(n_bytes)

  res := &rsa.PublicKey{}
  res.E = int(e)
  res.N = n

  return res
}

func (tv *TokenValidator) verifySignature(token string, key *rsa.PublicKey, alg string) bool {
  v := strings.Split(token, ".")
  method := jwt.GetSigningMethod(alg)
  if err := method.Verify(v[0]+"."+v[1], v[2], key); err != nil {
    log.WithError(err).Errorf("Verify Sign error. h: %s; p: %s", v[0], v[1])
    return false
  }
  return true
}

func jwtB64Decode(inp string, val any) error {
  inpBytes, err := base64.URLEncoding.DecodeString(inp)
  if err != nil {
    log.WithError(err).Errorf("Error in (in UrlEncoding) token-utils.jwtB64Encode.")
    return fmt.Errorf("500")
  }
  if err := json.Unmarshal(inpBytes, val); err != nil {
    log.WithError(err).Errorf("Error in (in unmarshalling) token-utils.jwtB64Encode.")
    return fmt.Errorf("500")
  }
  return nil
}

func (tv *TokenValidator) ParseAccessToken(token string) (any, any, string, error) {
  v := strings.Split(token, ".")

  header := &Header{}
  if err := jwtB64Decode(v[0], header); err != nil {
    log.Errorf("Error in parsing header.token-utils.parseToken. Input: %s", v[0])
```

```
    return nil, nil, "", fmt.Errorf("500")
  }

  payload := &AccessTokenPayload{}
  if err := jwtB64Decode(v[1], payload); err != nil {
    log.Errorf("Error in parsing payload.token-utils.parseToken. Input: %s", v[1])
    return nil, nil, "", fmt.Errorf("500")
  }

  return header, payload, v[2], nil
}

func (tv *TokenValidator) verifyAlg(header *Header) bool {
  return header.Alg == config.SigningAlg
}

func (tv *TokenValidator) verifyTyp(header *Header) bool {
  return header.Typ == config.TokenStandard
}

func (tv *TokenValidator) verifyTokenType(header *Header) bool {
  return header.TokenType == ACCESS
}

func (tv *TokenValidator) verifyIss(payload *AccessTokenPayload) bool {
  return payload.Iss == config.IdentityExterBaseUrl+"/"
}

func (tv *TokenValidator) verifyExp(payload *AccessTokenPayload) bool {
  now := time.Now().Unix()
  var leeway int64 = config.LeewaySeconds

  return payload.Exp+leeway >= now
}
```

Листинг 3.15: TokenValidator

```python
from pydantic import BaseModel
from uuid import UUID
from typing import Literal, List

class PongMessage(BaseModel):
    MessageType: str = 'PONG'

class AuthMessage(BaseModel):
    MessageType: str = 'AUTH'
    MessageId: int
    RoomUid: UUID
    Token: str
    LastEventId: int

class ActionMessage(BaseModel):
    MessageType: Literal['GAME-ACTION', 'VOTE']
    MessageId: int
    RoomUid: UUID
    UserUid: UUID
    ActionType: Literal['FOLD', 'CHECK', 'CALL', 'RAISE', 'OUTCOME'] | None = None
    Coef: Literal['X1_5', 'X2', 'ALL-IN'] | None = None
    VoteType: Literal['START', 'WAIT'] | None = None
```

Листинг 3.16: WS Messages from Player

```python
from pydantic import BaseModel
from typing import Literal

class SignUpRequest(BaseModel):
    scope: Literal['OPENID']
    username: str
    password: str

class AuthenticationRequest(BaseModel):
    scope: Literal['OPENID']
    grantType: Literal['PASSWORD', 'REFRESH-TOKEN']
    username: str | None = None
    password: str | None = None
    refreshToken: str | None = None
```

Листинг 3.17: Requests

```python
from pydantic import BaseModel
from uuid import UUID
from typing import Literal, List

class PlayingCard(BaseModel):
    cardSuit: Literal['DIAMONDS', 'HEARTS', 'CLUBS', 'SPADES']
    index: Literal['2', '3', '4', '5', '6', '7', '8', '9', '10', 'ACE', 'KING', 'QUEEN', 'JACK']

class UserInfo(BaseModel):
    userUid: UUID
    username: str
    numOfGames: int
    numOfWins: int
    userRank: Literal['РЕКРУТ', 'РЯДОВОЙ', 'СЕРЖАНТ', 'КАПИТАН', 'МАЙОР', 'ПОЛКОВНИК', 'ГЕНЕРАЛ']
    userState: Literal['IN-GAME', 'MENU']
    roomUid: UUID

class PlayerInfo(BaseModel):
    userUid: UUID
    username: str
    bet: int
    deposit: int
    lastActionLabel: Literal['NONE', 'FOLD', 'CHECK', 'CALL', 'RAISE', 'ALL-IN']
    userRank: Literal['РЕКРУТ', 'РЯДОВОЙ', 'СЕРЖАНТ', 'КАПИТАН', 'МАЙОР', 'ПОЛКОВНИК', 'ГЕНЕРАЛ']
    personalCardList: List[PlayingCard]

class RoomInfo(BaseModel):
    roomUid: UUID
    roomState: Literal['FORMING', 'GAMING', 'DISSOLUTION']
    playerList: List[PlayerInfo]
    tableCardList: List[PlayingCard]
    stack: int
    bout: UUID
    lastEventId: int
    roundNumber: int
```

Листинг 3.18: Info

```
from tkinter import *
from tkinter import messagebox
from tkinter import font
from PIL import ImageTk, Image
import requests

from color import FIELD_COLOR, FONT_COLOR

from poker import poker_table

from schemas.request import AuthenticationRequest, SignUpRequest
from config import BASE_URL, TOKENS

def pre_authorize(root, login, password, password2):

    if password != password2:
        messagebox.showerror('Password Error', 'Введенные пароли не совпадают!')
        return

    signup_request = SignUpRequest(scope='OPENID', username=login, password=password).model_dump_json()

    # print("\nЗапрос регистрации")
    # print(signup_request)

    signup_response = requests.post(BASE_URL+'/poker/v1/register', signup_request)

    # print("\nОтвет на запрос регистрации")
    # print(signup_response.status_code)
    # print(signup_response.json())

    if signup_response.status_code == 400:
        messagebox.showerror('Error 400', 'Аккаунт с указанным логином уже существует!')
        return
    elif signup_response.status_code == 500:
        messagebox.showerror('Error 500', 'Внутренняя ошибка сервера!')
        return

    play(root, login, password)


def my_account(root):

    # print("\nЗапрос информации о пользователе...")

    user_info_response = requests.get(BASE_URL+'/poker/v1/me',
                                      headers={'Authorization':TOKENS[0]})

    # print("\nОтвет на запрос о получении данных авторизованного пользователя")
    # print(user_info_response.status_code)
    # print(user_info_response.json())

    if user_info_response.status_code == 401:
        messagebox.showerror('Error 401', 'Время действия AccessToken истекло!')
        return
    elif user_info_response.status_code == 500:
        messagebox.showerror('Error 500', 'Внутренняя ошибка сервера!')
        return

    user_info = user_info_response.json()

    # root.iconify()

    account = Toplevel()
    account.grab_set()

    width = account.winfo_screenwidth()
    height = account.winfo_screenheight()
    x = (width - 750) / 2
    y = (height - 440) / 2

    account.geometry('750x440+%d+%d' % (x, y))
    account.title('TEXAS HOLDEM')
    account.resizable(False, False)
    account.configure(background = FIELD_COLOR)
    account.iconbitmap("icon.ico")

    font_reg = font.Font(family="Century Gothic", size=14)
```

```
    font_reg_big = font.Font(family="Century Gothic", size=25, weight='bold')

    poker_screen = ImageTk.PhotoImage(Image.open("entry_screen.jpg"))
    screen = Label(account, image = poker_screen, bg = FIELD_COLOR)
    screen.image_ref = poker_screen
    screen.pack()
    screen.place(x = -5, y = -5)

    play_label = Label(account, text='TEXAS HOLDEM',
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = FONT_COLOR,
                       font=font_reg_big)
    play_label.place(x = 445, y = 40)

    player_label = Label(account, text='Имя:',
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = 'white',
                       font=font_reg)
    player_label.place(x = 410, y = 110)

    player_label = Label(account, text='Игр сыграно:',
                       anchor = 'n',
                       bg = FIELD_COLOR,
                       fg = 'white',
                       font=font_reg)
    player_label.place(x = 410, y = 160)

    player_label = Label(account, text='Всего побед:',
                       anchor = 'n',
                       bg = FIELD_COLOR,
                       fg = 'white',
                       font=font_reg)
    player_label.place(x = 410, y = 210)

    player_label = Label(account, text='Винрейт:',
                       anchor = 'n',
                       bg = FIELD_COLOR,
                       fg = 'white',
                       font=font_reg)
    player_label.place(x = 410, y = 260)

    player_label = Label(account, text='Ранг:',
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = 'white',
                       font=font_reg)
    player_label.place(x = 410, y = 310)

    player_label2 = Label(account, text=user_info['Username'],
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = FONT_COLOR,
                       font=font_reg_big)
    player_label2.place(x = 550, y = 110)

    player_label2 = Label(account, text=str(user_info['NumOfGames']),
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = FONT_COLOR,
                       font=font_reg_big)
    player_label2.place(x = 550, y = 160)

    player_label2 = Label(account, text=str(user_info['NumOfWins']),
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = FONT_COLOR,
                       font=font_reg_big)
    player_label2.place(x = 550, y = 210)

    if user_info['NumOfGames'] == 0:
        text = '0'
    else:
        text=str(int(user_info['NumOfWins']/user_info['NumOfGames']*100))

    player_label2 = Label(account, text=text + '%',
                       anchor = 'c',
                       bg = FIELD_COLOR,
                       fg = FONT_COLOR,
                       font=font_reg_big)
    player_label2.place(x = 550, y = 260)
```

```
    player_label2 = Label(account, text=str(user_info['UserRank']),
                    anchor = 'c',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
                    font=font_reg_big)
    player_label2.place(x = 550, y = 310)


def play(root, login, password):

    authorization_request = AuthenticationRequest(scope='OPENID',
                                                   grantType='PASSWORD',
                                                   username=login,
                                                   password=password).model_dump_json()

    # print("\n Запрос на авторизацию пользователя")
    # print(authorization_request)

    authorization_response = requests.post(BASE_URL + '/poker/v1/oauth/token',
                                           authorization_request)

    print("\n Ответ на запрос авторизации пользователя")
    print(authorization_response.status_code)
    print(authorization_response.json())

    if authorization_response.status_code == 401:
        messagebox.showerror('Error 401', 'Неверное имя пользователя или пароль!')
        return
    elif authorization_response.status_code == 500:
        messagebox.showerror('Error 500', 'Внутренняя ошибка сервера!')
        return

    accessToken = authorization_response.json()['AccessToken']
    refreshToken = authorization_response.json()['RefreshToken']
    userUid = authorization_response.json()['UserUid']

    TOKENS[0] = accessToken
    TOKENS[1] = refreshToken
    TOKENS[2] = userUid
    TOKENS[3] = login

    root.destroy()

    play = Tk()
    play.grab_set()

    width = play.winfo_screenwidth()
    height = play.winfo_screenheight()
    x = (width - 750) / 2
    y = (height - 440) / 2

    play.geometry('750x440+%d+%d' % (x, y))
    play.title('TEXAS HOLDEM')
    play.resizable(False, False)
    play.configure(background = FIELD_COLOR)
    play.iconbitmap("icon.ico")

    font_reg = font.Font(family="Century Gothic", size=14)
    font_reg_big = font.Font(family="Century Gothic", size=25, weight='bold')

    poker_screen = ImageTk.PhotoImage(Image.open("entry_screen.jpg"))
    screen = Label(play, image = poker_screen, bg = FIELD_COLOR)
    screen.image_ref = poker_screen
    screen.pack()
    screen.place(x = -5, y = -5)


    play_label = Label(play, text='TEXAS HOLDEM',
                    anchor = 'c',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
                    font=font_reg_big)
    play_label.place(x = 445, y = 40)

    play_btn = Button(play, text='Играть',
                        width=20,
                        height=1,
                        font=font_reg,
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        relief = RIDGE,
```

```
                        command=poker_table)
    play_btn.place(anchor = 'w', x = 450, y = 150)

    me_btn = Button(play, text='Мой аккаунт',
                        width=20,
                        height=1,
                        font=font_reg,
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        relief = RIDGE,
                        command=lambda:my_account(play))
    me_btn.place(anchor = 'w', x = 450, y = 225)

    info_btn = Button(play, text='Комбинации и ранги',
                        width=20,
                        height=1,
                        font=font_reg,
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        relief = RIDGE)
    info_btn.place(anchor = 'w', x = 450, y = 300)



def registration(root):
    root.destroy()

    reg = Tk()
    reg.grab_set()

    width = reg.winfo_screenwidth()
    height = reg.winfo_screenheight()
    x = (width - 750) / 2
    y = (height - 440) / 2

    reg.geometry('750x440+%d+%d' % (x, y))
    reg.title('TEXAS HOLDEM')
    reg.resizable(False, False)
    reg.configure(background = FIELD_COLOR)
    reg.iconbitmap("icon.ico")

    font_reg = font.Font(family="Century Gothic", size=14)
    font_reg_big = font.Font(family="Century Gothic", size=25, weight='bold')

    poker_screen = ImageTk.PhotoImage(Image.open("entry_screen.jpg"))
    screen = Label(reg, image = poker_screen, bg = FIELD_COLOR)
    screen.image_ref = poker_screen
    screen.pack()
    screen.place(x = -5, y = -5)

    reg_label = Label(reg, text='Регистрация',
                    anchor = 'c',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
                    font=font_reg_big)
    reg_label.place(x = 450, y = 40)

    login_label = Label(reg, text='Логин',
                    anchor = 'w',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
                    font=font_reg)
    login_label.place(x = 450, y = 100)

    login_entry = Entry(reg, width = 20, font = font_reg)
    login_entry.place(x = 452, y = 130)

    password_label = Label(reg, text='Пароль',
                    anchor = 'w',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
                    font=font_reg)
    password_label.place(x = 450, y = 160)

    password_entry = Entry(reg, width = 20, show = '*', font = font_reg)
    password_entry.place(x = 452, y = 190)

    password_label2 = Label(reg, text='Подтвердите пароль',
                    anchor = 'w',
                    bg = FIELD_COLOR,
                    fg = FONT_COLOR,
```

```
                    font=font_reg)
    password_label2.place(x = 450, y = 220)

    password_entry2 = Entry(reg, width = 20, show = '*', font = font_reg)
    password_entry2.place(x = 452, y = 250)

    reg_btn = Button(reg, text='Зарегистрироваться',
                         width=20,
                         height=1,
                         font=font_reg,
                         bg = FIELD_COLOR,
                         fg = FONT_COLOR,
                         relief = RIDGE,
                         command=lambda:pre_authorize(reg, login_entry.get(),
                                                      password_entry.get(),
                                                      password_entry2.get()))
    reg_btn.place(anchor = 'w', x = 450, y = 340)


def authorization():
    root = Tk()

    width = root.winfo_screenwidth()
    height = root.winfo_screenheight()
    x = (width - 750) / 2
    y = (height - 440) / 2

    root.geometry('750x440+%d+%d' % (x, y))
    root.title('TEXAS HOLDEM')
    root.resizable(False, False)
    root.configure(background = FIELD_COLOR)
    root.iconbitmap("icon.ico")

    font_reg = font.Font(family="Century Gothic", size=14)
    font_reg_big = font.Font(family="Century Gothic", size=25, weight='bold')

    poker_screen = ImageTk.PhotoImage(Image.open("entry_screen.jpg"))
    screen = Label(root, image = poker_screen, bg = FIELD_COLOR)
    screen.image_ref = poker_screen
    screen.pack()
    screen.place(x = -5, y = -5)

    enter_label = Label(text='Авторизация',
                        anchor = 'c',
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        font=font_reg_big)
    enter_label.place(x = 450, y = 40)

    login_label = Label(text='Логин',
                        anchor = 'w',
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        font=font_reg)
    login_label.place(x = 450, y = 100)

    login_entry = Entry(root, width = 20, font = font_reg)
    login_entry.place(x = 452, y = 130)

    password_label = Label(text='Пароль',
                        anchor = 'w',
                        bg = FIELD_COLOR,
                        fg = FONT_COLOR,
                        font=font_reg)
    password_label.place(x = 450, y = 160)

    password_entry = Entry(root, width = 20, show = '*', font = font_reg)
    password_entry.place(x = 452, y = 190)

    login_btn = Button(text='Войти',
                         width=20,
                         height=1,
                         font=font_reg,
                         bg = FIELD_COLOR,
                         fg = FONT_COLOR,
                         relief = RIDGE,
                         command=lambda:play(root, login_entry.get(), password_entry.get()))
    login_btn.place(anchor = 'w', x = 450, y = 260)

    reg_label = Label(text='Нет аккаунта?',
                        anchor = 'w',
                        bg = FIELD_COLOR,
```

```
                    fg = FONT_COLOR,
                    font=font_reg)
    reg_label.place(x = 450, y = 300)

    reg_btn = Button(text='Зарегистрироваться',
                         width=20,
                         height=1,
                         font=font_reg,
                         bg = FIELD_COLOR,
                         fg = FONT_COLOR,
                         relief = RIDGE,
                        command=lambda:registration(root))
    reg_btn.place(anchor = 'w', x = 450, y = 350)

    root.mainloop()

authorization()
```

Листинг 3.19: Authorization

```
from tkinter import *
from tkinter import messagebox
from tkinter import font
from PIL import ImageTk, Image

from time import time, sleep

from color import FIELD_COLOR, FIELD_COLOR2, FONT_COLOR, PLAYER_COL_COLOR
from config import BASE_URL, TOKENS, WS_BASE_URL

from schemas.ws_from_player import AuthMessage, ActionMessage, PongMessage

pong_msg = PongMessage().model_dump_json()

from threading import Thread
import requests
import websocket
import json

# ИГРОКИ
# players = [
#     ['sdkfs-dvwej-bwevw-eb', 'Player1', 'ПОЛКОВНИК', '500', '1250'],
#     ['Player2', 'РЯДОВОЙ', '5000', '2500'],
#     ['Player3', 'ГЕНЕРАЛ', '12500', '0'],
#     ['Player4', 'КАПИТАН', '7250', '0'],
# ]
player = ['', '', '', '', '', False, False]
players = []

# ОСНОВНАЯ ИНФОРМАЦИЯ
info = [
    '0',
    '10 000',
    '0',
    ''
]

auth = [True]
vote = [True, False]
action = ['']
delay = [False]

def poker_table():

    room_info_response = requests.get(BASE_URL+'/poker/v1/rooms/matching',
                                      headers={'Authorization':TOKENS[0]})

    if room_info_response.status_code == 401:
        messagebox.showerror('Error 401', 'Время действия AccessToken истекло!')
        return
    elif room_info_response.status_code == 500:
        messagebox.showerror('Error 500', 'Внутренняя ошибка сервера!')
        return

    room_info = room_info_response.json()
    room_uid = room_info['RoomUid']
    last_event_id = room_info['LastEventId']

    for room_player in room_info['PlayerList']:
        player[0] = room_player['UserUid']
        player[1] = room_player['Username']
```

```
        player[2] = room_player['UserRank']
        player[3] = str(room_player['Deposit'])
        player[4] = str(room_player['Bet'])
        print(player)
        player_c = player.copy()
        players.append(player_c)


poker_table = Toplevel()
# poker_table = Tk()
poker_table.grab_set()
poker_table.lift()

# ШРИФТЫ
font_player = font.Font(family="Century Gothic", size=20, weight='bold')
font_bet = font.Font(family="Century Gothic", size=20)
font_bet_big = font.Font(family="Century Gothic", size=30, weight='bold')

width = poker_table.winfo_screenwidth() - 10
height = poker_table.winfo_screenheight() - 30

poker_table.iconbitmap("icon.ico")
poker_table.geometry('%dx%d+0+0' % (width, height))
poker_table.title('TEXAS HOLDEM -- '+TOKENS[3])
poker_table.resizable(True, True)
poker_table.configure(background = FIELD_COLOR2)

table_img = PhotoImage(file="img/table.png")
table = Label(poker_table, image=table_img, bg=FIELD_COLOR2)
table.place(x= 285, y= 20)

bet_img = PhotoImage(file="img/money.png")
bet_label = Label(poker_table, image=bet_img, bg=FIELD_COLOR2)
bet_label.place(x= 666, y= 75)

# РАНГИ
recruit_img = PhotoImage(file="img/ranks/recruit.png")
soldier_img = PhotoImage(file="img/ranks/soldier.png")
sergeant_img = PhotoImage(file="img/ranks/sergeant.png")
captain_img = PhotoImage(file="img/ranks/captain.png")
major_img = PhotoImage(file="img/ranks/major.png")
colonel_img = PhotoImage(file="img/ranks/colonel.png")
general_img = PhotoImage(file="img/ranks/general.png")

ranks_dict = {
    'РЕКРУТ': recruit_img,
    'РЯДОВОЙ': soldier_img,
    'СЕРЖАНТ': sergeant_img,
    'КАПИТАН': captain_img,
    'МАЙОР': major_img,
    'ПОЛКОВНИК': colonel_img,
    'ГЕНЕРАЛ': general_img
}

# КАРТЫ
clubs_2_img = PhotoImage(file="img/pack/CLUBS/2.png")
clubs_3_img = PhotoImage(file="img/pack/CLUBS/3.png")
clubs_4_img = PhotoImage(file="img/pack/CLUBS/4.png")
clubs_5_img = PhotoImage(file="img/pack/CLUBS/5.png")
clubs_6_img = PhotoImage(file="img/pack/CLUBS/6.png")
clubs_7_img = PhotoImage(file="img/pack/CLUBS/7.png")
clubs_8_img = PhotoImage(file="img/pack/CLUBS/8.png")
clubs_9_img = PhotoImage(file="img/pack/CLUBS/9.png")
clubs_10_img = PhotoImage(file="img/pack/CLUBS/10.png")
clubs_jack_img = PhotoImage(file="img/pack/CLUBS/JACK.png")
clubs_queen_img = PhotoImage(file="img/pack/CLUBS/QUEEN.png")
clubs_king_img = PhotoImage(file="img/pack/CLUBS/KING.png")
clubs_ace_img = PhotoImage(file="img/pack/CLUBS/ACE.png")

diamonds_2_img = PhotoImage(file="img/pack/DIAMONDS/2.png")
diamonds_3_img = PhotoImage(file="img/pack/DIAMONDS/3.png")
diamonds_4_img = PhotoImage(file="img/pack/DIAMONDS/4.png")
diamonds_5_img = PhotoImage(file="img/pack/DIAMONDS/5.png")
diamonds_6_img = PhotoImage(file="img/pack/DIAMONDS/6.png")
diamonds_7_img = PhotoImage(file="img/pack/DIAMONDS/7.png")
diamonds_8_img = PhotoImage(file="img/pack/DIAMONDS/8.png")
diamonds_9_img = PhotoImage(file="img/pack/DIAMONDS/9.png")
diamonds_10_img = PhotoImage(file="img/pack/DIAMONDS/10.png")
diamonds_jack_img = PhotoImage(file="img/pack/DIAMONDS/JACK.png")
diamonds_queen_img = PhotoImage(file="img/pack/DIAMONDS/QUEEN.png")
diamonds_king_img = PhotoImage(file="img/pack/DIAMONDS/KING.png")
diamonds_ace_img = PhotoImage(file="img/pack/DIAMONDS/ACE.png")
```

```
hearts_2_img = PhotoImage(file="img/pack/HEARTS/2.png")
hearts_3_img = PhotoImage(file="img/pack/HEARTS/3.png")
hearts_4_img = PhotoImage(file="img/pack/HEARTS/4.png")
hearts_5_img = PhotoImage(file="img/pack/HEARTS/5.png")
hearts_6_img = PhotoImage(file="img/pack/HEARTS/6.png")
hearts_7_img = PhotoImage(file="img/pack/HEARTS/7.png")
hearts_8_img = PhotoImage(file="img/pack/HEARTS/8.png")
hearts_9_img = PhotoImage(file="img/pack/HEARTS/9.png")
hearts_10_img = PhotoImage(file="img/pack/HEARTS/10.png")
hearts_jack_img = PhotoImage(file="img/pack/HEARTS/JACK.png")
hearts_queen_img = PhotoImage(file="img/pack/HEARTS/QUEEN.png")
hearts_king_img = PhotoImage(file="img/pack/HEARTS/KING.png")
hearts_ace_img = PhotoImage(file="img/pack/HEARTS/ACE.png")

spades_2_img = PhotoImage(file="img/pack/SPADES/2.png")
spades_3_img = PhotoImage(file="img/pack/SPADES/3.png")
spades_4_img = PhotoImage(file="img/pack/SPADES/4.png")
spades_5_img = PhotoImage(file="img/pack/SPADES/5.png")
spades_6_img = PhotoImage(file="img/pack/SPADES/6.png")
spades_7_img = PhotoImage(file="img/pack/SPADES/7.png")
spades_8_img = PhotoImage(file="img/pack/SPADES/8.png")
spades_9_img = PhotoImage(file="img/pack/SPADES/9.png")
spades_10_img = PhotoImage(file="img/pack/SPADES/10.png")
spades_jack_img = PhotoImage(file="img/pack/SPADES/JACK.png")
spades_queen_img = PhotoImage(file="img/pack/SPADES/QUEEN.png")
spades_king_img = PhotoImage(file="img/pack/SPADES/KING.png")
spades_ace_img = PhotoImage(file="img/pack/SPADES/ACE.png")

cards_dict = {
    'CLUBS': {
        '2': clubs_2_img,
        '3': clubs_3_img,
        '4': clubs_4_img,
        '5': clubs_5_img,
        '6': clubs_6_img,
        '7': clubs_7_img,
        '8': clubs_8_img,
        '9': clubs_9_img,
        '10': clubs_10_img,
        'JACK': clubs_jack_img,
        'QUEEN': clubs_queen_img,
        'KING': clubs_king_img,
        'ACE': clubs_ace_img
    },
    'DIAMONDS': {
        '2': diamonds_2_img,
        '3': diamonds_3_img,
        '4': diamonds_4_img,
        '5': diamonds_5_img,
        '6': diamonds_6_img,
        '7': diamonds_7_img,
        '8': diamonds_8_img,
        '9': diamonds_9_img,
        '10': diamonds_10_img,
        'JACK': diamonds_jack_img,
        'QUEEN': diamonds_queen_img,
        'KING': diamonds_king_img,
        'ACE': diamonds_ace_img
    },
    'HEARTS': {
        '2': hearts_2_img,
        '3': hearts_3_img,
        '4': hearts_4_img,
        '5': hearts_5_img,
        '6': hearts_6_img,
        '7': hearts_7_img,
        '8': hearts_8_img,
        '9': hearts_9_img,
        '10': hearts_10_img,
        'JACK': hearts_jack_img,
        'QUEEN': hearts_queen_img,
        'KING': hearts_king_img,
        'ACE': hearts_ace_img
    },
    'SPADES': {
        '2': spades_2_img,
        '3': spades_3_img,
        '4': spades_4_img,
        '5': spades_5_img,
        '6': spades_6_img,
        '7': spades_7_img,
```

```
        '8': spades_8_img,
        '9': spades_9_img,
        '10': spades_10_img,
        'JACK': spades_jack_img,
        'QUEEN': spades_queen_img,
        'KING': spades_king_img,
        'ACE': spades_ace_img
    }
}

cover_img = PhotoImage(file="img/pack/cover.png")
cover_img_mini = PhotoImage(file="img/pack/cover_mini.png")

# КНОПКИ
call_img = PhotoImage(file="img/buttons/call.png")
a_call_img = PhotoImage(file="img/buttons/call_a.png")
check_img = PhotoImage(file="img/buttons/check.png")
a_check_img = PhotoImage(file="img/buttons/check_a.png")
fold_img = PhotoImage(file="img/buttons/fold.png")
a_fold_img = PhotoImage(file="img/buttons/fold_a.png")
raise1_img = PhotoImage(file="img/buttons/raise1.png")
a_raise1_img = PhotoImage(file="img/buttons/raise1_a.png")
raise2_img = PhotoImage(file="img/buttons/raise2.png")
a_raise2_img = PhotoImage(file="img/buttons/raise2_a.png")
allin_img = PhotoImage(file="img/buttons/allin.png")
a_allin_img = PhotoImage(file="img/buttons/allin_a.png")
wait_img = PhotoImage(file="img/buttons/wait.png")
begin_img = PhotoImage(file="img/buttons/begin.png")

# ЛЕЙБЛЫ
    # КАРТЫ
tcard1 = Label(poker_table, bg=FIELD_COLOR2)
tcard2 = Label(poker_table, bg=FIELD_COLOR2)
tcard3 = Label(poker_table, bg=FIELD_COLOR2)
tcard4 = Label(poker_table, bg=FIELD_COLOR2)
tcard5 = Label(poker_table, bg=FIELD_COLOR2)

pcard1 = Label(poker_table, bg=FIELD_COLOR2)
pcard2 = Label(poker_table, bg=FIELD_COLOR2)

tcard1.place(x= 365, y= 145)
tcard2.place(x= 545, y= 145)
tcard3.place(x= 725, y= 145)
tcard4.place(x= 905, y= 145)
tcard5.place(x= 1085, y= 145)

pcard1.place(x= 455, y= 525)
pcard2.place(x= 635, y= 525)

    # КНОПКИ
def on_enter_call(e):
    call_btn['image'] = a_call_img
def on_leave_call(e):
    call_btn['image'] = call_img

def on_enter_check(e):
    check_btn['image'] = a_check_img
def on_leave_check(e):
    check_btn['image'] = check_img

def on_enter_fold(e):
    fold_btn['image'] = a_fold_img
def on_leave_fold(e):
    fold_btn['image'] = fold_img

def on_enter_raise1(e):
    raise1_btn['image'] = a_raise1_img
def on_leave_raise1(e):
    raise1_btn['image'] = raise1_img

def on_enter_raise2(e):
    raise2_btn['image'] = a_raise2_img
def on_leave_raise2(e):
    raise2_btn['image'] = raise2_img

def on_enter_allin(e):
    allin_btn['image'] = a_allin_img
def on_leave_allin(e):
    allin_btn['image'] = allin_img

def on_disable(e):
    pass
```

```
def disable(btns, btn_type):
    action[0] = btn_type
    for btn in btns:
        btn['state'] = DISABLED
        btn.bind("<Enter>", on_disable)

def vote_action():
    vote[1]=True
    if vote[0]:
        vote_btn['image'] = wait_img
        vote[0]=False
        return
    vote_btn['image'] = begin_img
    vote[0]=True


call_btn = Button(poker_table,
                  image=call_img,
                  bg=FIELD_COLOR2,
                  activebackground=FIELD_COLOR2,
                  relief = FLAT, bd=0,
                  state=DISABLED,
                  command=lambda:disable([call_btn,
                                          check_btn,
                                          fold_btn,
                                          raise1_btn,
                                          raise2_btn,
                                          allin_btn], 'call'))
# call_btn.bind("<Enter>", on_enter_call)
# call_btn.bind("<Leave>", on_leave_call)
call_btn.place(x=850, y= 520)

check_btn = Button(poker_table,
                  image=check_img,
                  bg=FIELD_COLOR2,
                  activebackground=FIELD_COLOR2,
                  relief = FLAT, bd=0,
                  state=DISABLED,
                  command=lambda:disable([call_btn,
                                          check_btn,
                                          fold_btn,
                                          raise1_btn,
                                          raise2_btn,
                                          allin_btn], 'check'))

# check_btn.bind("<Enter>", on_enter_check)
# check_btn.bind("<Leave>", on_leave_check)
check_btn.place(x=850, y= 590)

fold_btn = Button(poker_table,
                  image=fold_img,
                  bg=FIELD_COLOR2,
                  activebackground=FIELD_COLOR2,
                  relief = FLAT, bd=0,
                  state=DISABLED,
                  command=lambda:disable([call_btn,
                                          check_btn,
                                          fold_btn,
                                          raise1_btn,
                                          raise2_btn,
                                          allin_btn], 'fold'))
# fold_btn.bind("<Enter>", on_enter_fold)
# fold_btn.bind("<Leave>", on_leave_fold)
fold_btn.place(x=850, y= 660)

raise1_btn = Button(poker_table,
                  image=raise1_img,
                  bg=FIELD_COLOR2,
                  activebackground=FIELD_COLOR2,
                  relief = FLAT, bd=0,
                  state=DISABLED,
                  command=lambda:disable([call_btn,
                                          check_btn,
                                          fold_btn,
                                          raise1_btn,
                                          raise2_btn,
                                          allin_btn], 'raise1'))
# raise1_btn.bind("<Enter>", on_enter_raise1)
# raise1_btn.bind("<Leave>", on_leave_raise1)
raise1_btn.place(x=1020, y= 520)
```

```
raise2_btn = Button(poker_table,
                    image=raise2_img,
                    bg=FIELD_COLOR2,
                    activebackground=FIELD_COLOR2,
                    relief = FLAT, bd=0,
                    state=DISABLED,
                    command=lambda:disable([call_btn,
                                            check_btn,
                                            fold_btn,
                                            raise1_btn,
                                            raise2_btn,
                                            allin_btn], 'raise2'))
# raise2_btn.bind("<Enter>", on_enter_raise2)
# raise2_btn.bind("<Leave>", on_leave_raise2)
raise2_btn.place(x=1020, y= 590)

allin_btn = Button(poker_table,
                    image=allin_img,
                    bg=FIELD_COLOR2,
                    activebackground=FIELD_COLOR2,
                    relief = FLAT, bd=0,
                    state=DISABLED,
                    command=lambda:disable([call_btn,
                                            check_btn,
                                            fold_btn,
                                            raise1_btn,
                                            raise2_btn,
                                            allin_btn], 'allin'))
# allin_btn.bind("<Enter>", on_enter_allin)
# allin_btn.bind("<Leave>", on_leave_allin)
allin_btn.place(x=1020, y= 660)

vote_btn = Button(poker_table,
                    image=begin_img,
                    bg=FIELD_COLOR2,
                    activebackground=FIELD_COLOR2,
                    relief = FLAT, bd=0,
                    command=vote_action)
vote_btn.place(x=20, y= 660)


    # ИНФОРМАЦИЯ
bank_label = Label(poker_table,
        anchor = 'c',
        bg = FIELD_COLOR2,
        fg = 'white',
        font=font_bet_big)
bank_label.place(x = 750, y = 80)

capital_label = Label(poker_table,
        anchor = 'c',
        bg = FIELD_COLOR2,
        fg = 'white',
        font=font_bet)
capital_label.place(x = 860, y = 430)

bet_label = Label(poker_table,
        anchor = 'c',
        bg = FIELD_COLOR2,
        fg = 'white',
        font=font_bet)
bet_label.place(x = 1035, y = 430)

combination_label = Label(poker_table,
        anchor = 'c',
        bg = FIELD_COLOR2,
        fg = 'white',
        font=font_player)
combination_label.place(x = 470, y = 430)

players_col = Canvas(poker_table, bg=FIELD_COLOR2, width=255, height=height-150,highlightthickness=0)
players_col.pack(anchor=NW, expand=1)

def show_players():
    shift = 0
    for table_player in players:
        fgc='white'
        if table_player[6]:
            fgc='gray60'

        players_col.create_rectangle(5, shift + 5, 250, shift + 125, outline=fgc)
```

```
            player_label = Label(poker_table,
                    text=table_player[1],
                    anchor = 'c',
                    bg = FIELD_COLOR2,
                    fg = fgc,
                    font=font_player)
            player_label.place(x = 6, y = shift + 6)

            player_label2 = Label(poker_table,
                            text='00000000',
                            anchor = 'c',
                            bg = FIELD_COLOR2,
                            fg = FIELD_COLOR2,
                            font=font_bet)
            player_label2.place(x = 6, y = shift + 85)

            player_label2 = Label(poker_table,
                            text=table_player[3],
                            anchor = 'c',
                            bg = FIELD_COLOR2,
                            fg = fgc,
                            font=font_bet)
            player_label2.place(x = 6, y = shift + 85)

            player_label3 = Label(poker_table,
                            text='0000000',
                            anchor = 'c',
                            bg = FIELD_COLOR2,
                            fg = FIELD_COLOR2,
                            font=font_bet)
            player_label3.place(x = 135, y = shift + 85)

            player_label3 = Label(poker_table,
                            text=table_player[4],
                            anchor = 'c',
                            bg = FIELD_COLOR2,
                            fg = fgc,
                            font=font_bet)
            player_label3.place(x = 135, y = shift + 85)

            rank_img = ranks_dict[table_player[2]]

            rank_label = Label(poker_table, image=rank_img, bg=FIELD_COLOR2)
            rank_label.place(x= 10, y= shift + 50)

            if table_player[5]:

                pcard1_label = Label(poker_table, image=cover_img_mini,
                        bg = FIELD_COLOR2)
                pcard1_label.place(x = 140, y = shift + 10)

                pcard2_label = Label(poker_table, image=cover_img_mini,
                        bg = FIELD_COLOR2)
                pcard2_label.place(x = 195, y = shift + 10)

            shift += 125

    def show_info():
        bank_label['text'] = info[0]
        capital_label['text'] = info[1]
        bet_label['text'] = info[2]
        combination_label['text'] = info[3]

    def get_player_info(user_uid):
        response = requests.get(BASE_URL+'/poker/v1/players/' + user_uid,
                                headers={'Authorization':TOKENS[0]})
        print(response)

        if response.status_code == 200:
            player_info = response.json()

            player[0] = player_info['UserUid']
            player[1] = player_info['Username']
            player[2] = player_info['UserRank']
            player[3] = '10000'
            player[4] = '0'
            player_c = player.copy()

            players.append(player_c)
            show_players()

    def winner_result(winners, combo):
```

```
    messagebox.showwarning('WINNER(S)', 'Поебедитель(-и):' + winners
                                    + '\nКомбинация: ' + combo, parent=poker_table)

def check_delay():
    sleep(3)



show_players()
show_info()

def on_message(ws, message):
    message = json.loads(message)

    # delay[0] = True
    # tp=Thread(target=lambda:chek_delay())
    # tp.start()


    if message['MessageType'] == 'PING':
        ws.send(pong_msg)

        if auth[0]:
            auth_msg = AuthMessage(MessageId=int(time()*1000),
                    RoomUid=room_uid,
                    Token=TOKENS[0],
                    LastEventId=last_event_id
                    ).model_dump_json()
            ws.send(auth_msg)
            auth[0] = False

        if action[0] != '':
            if action[0] == 'fold':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='FOLD'
                        ).model_dump_json()
                ws.send(act_msg)
                action[0] = ''

            elif action[0] == 'check':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='CHECK'
                        ).model_dump_json()
                ws.send(act_msg)
                action[0] = ''

            elif action[0] == 'call':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='CALL'
                        ).model_dump_json()
                ws.send(act_msg)
                action[0] = ''

            elif action[0] == 'raise1':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='RAISE',
                        Coef='X1_5'
                        ).model_dump_json()
                ws.send(act_msg)
                action[0] = ''

            elif action[0] == 'raise2':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='RAISE',
                        Coef='X2'
                        ).model_dump_json()
                ws.send(act_msg)
```

```
                action[0] = ''

            elif action[0] == 'allin':
                act_msg = ActionMessage(MessageType='GAME-ACTION',
                        MessageId=int(time()*1000),
                        RoomUid=room_uid,
                        UserUid=TOKENS[2],
                        ActionType='RAISE',
                        Coef='ALL-IN'
                        ).model_dump_json()
                ws.send(act_msg)
                action[0] = ''

        if vote[1]:
            vote[1] = False
            if vote[0]:
                act_msg = ActionMessage(MessageType='VOTE',
                    MessageId=int(time()*1000),
                    RoomUid=room_uid,
                    UserUid=TOKENS[2],
                    VoteType='WAIT'
                    ).model_dump_json()
                ws.send(act_msg)
            else:
                act_msg = ActionMessage(MessageType='VOTE',
                    MessageId=int(time()*1000),
                    RoomUid=room_uid,
                    UserUid=TOKENS[2],
                    VoteType='START'
                    ).model_dump_json()
                ws.send(act_msg)



    elif message['MessageType'] == 'EVENT':

        if message['EventType'] == 'PLAYER-ACTION-EVENT':

            user_uid = message['EventDescriptor']['UserUid']

            if message['EventDescriptor']['ActionType'] == 'INCOME':
                f = False
                for table_player in players:
                    if user_uid != table_player[0]:
                        f = True

                if f:
                    tp=Thread(target=lambda:get_player_info(user_uid))
                    tp.start()

            elif message['EventDescriptor']['ActionType'] == 'OUTCOME':
                for table_player in players:
                    if user_uid == table_player[0]:
                        table_player[6] = True
                        show_players()
                if user_uid == TOKENS[2]:
                    messagebox.showerror('YOU LOOSE', 'К сожалению, вы проиграли!', parent=poker_table)
                    # poker_table.destroy()

            elif message['EventDescriptor']['ActionType'] == 'BOUT':
                if user_uid == TOKENS[2]:
                    info[3] = message['EventDescriptor']['BestCombName']
                    for bout_variant in message['EventDescriptor']['BoutVariants']:

                        # РАЗБЛОКИРОВАТЬ КНОПКИ
                        if bout_variant['VariantType'] == 'FOLD':
                            fold_btn.config(state=NORMAL)
                            fold_btn.bind("<Enter>", on_enter_fold)
                            fold_btn.bind("<Leave>", on_leave_fold)
                        elif bout_variant['VariantType'] == 'CHECK':
                            check_btn.config(state=NORMAL)
                            check_btn.bind("<Enter>", on_enter_check)
                            check_btn.bind("<Leave>", on_leave_check)
                        elif bout_variant['VariantType'] == 'CALL':
                            call_btn.config(state=NORMAL)
                            call_btn.bind("<Enter>", on_enter_call)
                            call_btn.bind("<Leave>", on_leave_call)
                        elif bout_variant['VariantType'] == 'RAISE':

                            for raise_variant in bout_variant['RaiseVariants']:
                                if raise_variant == 'X1_5':
                                    raise1_btn.config(state=NORMAL)
```

```
                        raise1_btn.bind("<Enter>", on_enter_raise1)
                        raise1_btn.bind("<Leave>", on_leave_raise1)
                    elif raise_variant == 'X2':
                        raise2_btn.config(state=NORMAL)
                        raise2_btn.bind("<Enter>", on_enter_raise2)
                        raise2_btn.bind("<Leave>", on_leave_raise2)
                    elif raise_variant == 'ALL-IN':
                        allin_btn.config(state=NORMAL)
                        allin_btn.bind("<Enter>", on_enter_allin)
                        allin_btn.bind("<Leave>", on_leave_allin)

            show_info()

        elif message['EventDescriptor']['ActionType'] == 'FOLD':
            for table_player in players:
                if table_player[0] == user_uid:
                    table_player[6] = True
            show_players()
            if user_uid == TOKENS[2]:
                disable([call_btn, check_btn, fold_btn,
                         raise1_btn, raise2_btn, allin_btn], '')


        elif message['EventDescriptor']['ActionType'] == 'CHECK':
            if user_uid == TOKENS[2]:
                disable([call_btn, check_btn, fold_btn,
                         raise1_btn, raise2_btn, allin_btn], '')

        elif message['EventDescriptor']['ActionType'] == 'CALL':
            for table_player in players:
                if table_player[0] == user_uid:
                    table_player[3] = str(message['EventDescriptor']['NewDeposit'])
                    table_player[4] = str(message['EventDescriptor']['NewBet'])
            if user_uid == TOKENS[2]:
                info[1] = str(message['EventDescriptor']['NewDeposit'])
                info[2] = str(message['EventDescriptor']['NewBet'])
                disable([call_btn, check_btn, fold_btn,
                         raise1_btn, raise2_btn, allin_btn], '')
            show_players()
            show_info()

        elif message['EventDescriptor']['ActionType'] == 'RAISE':
            for table_player in players:
                if table_player[0] == user_uid:
                    table_player[3] = str(message['EventDescriptor']['NewDeposit'])
                    table_player[4] = str(message['EventDescriptor']['NewBet'])
            if user_uid == TOKENS[2]:
                info[1] = str(message['EventDescriptor']['NewDeposit'])
                info[2] = str(message['EventDescriptor']['NewBet'])
                disable([call_btn, check_btn, fold_btn,
                         raise1_btn, raise2_btn, allin_btn], '')
            show_players()
            show_info()

        elif message['EventDescriptor']['ActionType'] == 'ALL-IN':
            for table_player in players:
                if table_player[0] == user_uid:
                    table_player[3] = str(message['EventDescriptor']['NewDeposit'])
                    table_player[4] = str(message['EventDescriptor']['NewBet'])
            if user_uid == TOKENS[2]:
                info[1] = str(message['EventDescriptor']['NewDeposit'])
                info[2] = str(message['EventDescriptor']['NewBet'])
                disable([call_btn, check_btn, fold_btn,
                         raise1_btn, raise2_btn, allin_btn], '')
            show_players()
            show_info()

        elif message['EventDescriptor']['ActionType'] == 'SET-DEALER':
            pass

        elif message['EventDescriptor']['ActionType'] == 'MIN-BLIND-IN':
            print()
            for table_player in players:
                print(table_player)
                if table_player[0] == user_uid:
                    table_player[3] = str(message['EventDescriptor']['NewDeposit'])
                    table_player[4] = str(message['EventDescriptor']['NewBet'])
                    print(table_player)
            print()
            if user_uid == TOKENS[2]:
                info[1] = str(message['EventDescriptor']['NewDeposit'])
                info[2] = str(message['EventDescriptor']['NewBet'])
```

```
            show_players()
            show_info()

        elif message['EventDescriptor']['ActionType'] == 'MAX-BLIND-IN':
            for table_player in players:
                if table_player[0] == user_uid:
                    table_player[3] = str(message['EventDescriptor']['NewDeposit'])
                    table_player[4] = str(message['EventDescriptor']['NewBet'])
            if user_uid == TOKENS[2]:
                info[1] = str(message['EventDescriptor']['NewDeposit'])
                info[2] = str(message['EventDescriptor']['NewBet'])
            show_players()
            show_info()

    elif message['EventType'] == 'GAME-EVENT':

        if message['EventDescriptor']['EventType'] == 'ROOM_STATE_UPDATE':
            if message['EventDescriptor']['NewRoomState'] == 'GAMING':
                vote_btn.destroy()
                # tcard1.config(image=cover_img)
                # tcard2.config(image=cover_img)
                # tcard3.config(image=cover_img)
                # tcard4.config(image=cover_img)
                # tcard5.config(image=cover_img)
                # pcard1.config(image=cover_img)
                # pcard2.config(image=cover_img)
            elif message['EventDescriptor']['NewRoomState'] == 'DISSLOLUTION':
                messagebox.showerror('YOU WIN', 'Поздравляем, вы победили!')
                ws.close()
                poker_table.destroy()

        elif message['EventDescriptor']['EventType'] == 'NEW_ROUND':
            tcard1.config(image=cover_img)
            tcard2.config(image=cover_img)
            tcard3.config(image=cover_img)
            tcard4.config(image=cover_img)
            tcard5.config(image=cover_img)
            pcard1.config(image=cover_img)
            pcard2.config(image=cover_img)
            for table_player in players:
                table_player[5] = True
                table_player[6] = False

        elif message['EventDescriptor']['EventType'] == 'NEW_TRADE_ROUND':
            pass

        elif message['EventDescriptor']['EventType'] == 'PERSONAL_CARDS':
            suit1 = message['EventDescriptor']['PlayingCardsList'][0]['CardSuit']
            index1 = message['EventDescriptor']['PlayingCardsList'][0]['Index']
            suit2 = message['EventDescriptor']['PlayingCardsList'][1]['CardSuit']
            index2 = message['EventDescriptor']['PlayingCardsList'][1]['Index']

            pcard1.config(image=cards_dict[suit1][index1])
            pcard2.config(image=cards_dict[suit2][index2])

            info[3] = message['EventDescriptor']['BestCombName']
            show_info()


        elif message['EventDescriptor']['EventType'] == 'CARDS_ON_TABLE':
            i = 1
            for card in message['EventDescriptor']['PlayingCardsList']:
                suit = card['CardSuit']
                index = card['Index']
                if i == 1:
                    tcard1.config(image=cards_dict[suit][index])
                elif i == 2:
                    tcard2.config(image=cards_dict[suit][index])
                elif i == 3:
                    tcard3.config(image=cards_dict[suit][index])
                elif i == 4:
                    tcard4.config(image=cards_dict[suit][index])
                elif i == 5:
                    tcard5.config(image=cards_dict[suit][index])
                i+=1

            info[3] = message['EventDescriptor']['BestCombName']
            show_info()


        elif message['EventDescriptor']['EventType'] == 'BET_ACCEPTED':
```

```
                    for table_player in players:
                        table_player[4] = ''
                    info[0] = str(message['EventDescriptor']['NewStack'])
                    show_players()
                    show_info()


                elif message['EventDescriptor']['EventType'] == 'WINNER_RESULT':
                    winners = ''
                    i = 0
                    for uuid in message['EventDescriptor']['WinnerUids']:
                        for table_player in players:
                            if uuid == table_player[0]:
                                table_player[3] == str(message['EventDescriptor']['WinnerDeposits'][i])
                                i+=1
                                winners+= ' ' + table_player[1]
                    combo = message['EventDescriptor']['BestCombName']


                    info[0] = '0'

                    show_players()
                    show_info()

                    tp=Thread(target=lambda:winner_result(winners, combo))
                    tp.start()

    def on_open(ws):
        print("##### opened #####")

    websocket.enableTrace(True)
    ws = websocket.WebSocketApp(WS_BASE_URL
                                + 'poker/v1/rooms-ws/'
                                + room_uid + '?uid=' +  TOKENS[2],
                                on_message = on_message,
                                on_open= on_open)
    wst = Thread(target=ws.run_forever)
    wst.daemon = True
    wst.start()

    poker_table.mainloop()

# poker_table()
```

Листинг 3.20: PokerGame