Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРДЖАЮ
Заведующий кафедрой
_____ О. В. Непомнящий
подпись
«_____» _____ 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 — Информатика и вычислительная техника

Игра «Монополия». Серверная часть

Руководитель	_____ подпись, дата	ст. преподаватель	В. С. Васильев
Выпускник	_____ подпись, дата		С. А. Звягин
Нормоконтролер	_____ подпись, дата	ст. преподаватель	В. С. Васильев

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Игра «Монополия». Серверная часть» содержит 70 страниц текстового документа, 25 иллюстрации, 10 таблиц, 5 приложений, 17 использованных источников.

Ключевые слова: СЕРВЕРНОЕ ПРИЛОЖЕНИЕ, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА, ПРОЕКТИРОВАНИЕ СЕРВЕРНЫХ СИСТЕМ, РАЗРАБОТКА ИГР.

Цель работы: программная реализация серверной части игры «Монополия».

Выпускная квалификационная работа состоит из введения, основной части из трёх глав и заключения. Структура работы отражает решаемые задачи.

Во введении определяется цель работы, на её основе выделяются задачи.

В первой главе рассматриваются аналоги системы, формируется спецификация требований к разрабатываемой система на основе прецедентов.

Во второй главе определяются основные сущности предметной области, определяется тип используемой архитектуры серверного программного обеспечения, описывается и демонстрируются с помощью диаграмм интерактивный процесс проектирования, формируется формат клиент-серверного взаимодействия.

В третьей главе описываются применённые в ходе разработки инструменты, приводится описание процессов тестирования и развёртывания серверной системы.

В заключении формулируются основные итоги по выполненной работе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

«Монополия» — одна из самых известных настольных игр, концепция которой не нуждается в представлении. **Целью работы** является программная клиент-серверная реализация игры по мотивам классической «Монополии», а в частности, её серверная часть. Серверная часть должна отвечать требованиям гибкости в изменении игровых условий, высокой отказоустойчивости и масштабируемости.

Решаемые **задачи** отражены в структуре работы:

Первая глава посвящена рассмотрению существующих аналогов в предметной области, анализу слабых и сильных сторон существующих реализаций, описанию игрового процесса. На основании этого в первой главе формируется спецификация требований к серверному приложению.

Вторая глава посвящена разработке архитектуры системы, структуры базы данных и алгоритмам обработки игровых механик.

Третья глава посвящена описанию процессов и средств разработки и тестирования, в ней так же приводится информация о сборке и развёртыванию серверной системы.

1 Разработка спецификации требований

Разрабатываемая игра «Монополия» содержит отхождения от правил классической игры и её игровых механик, за счёт этого формируются конкурентные преимущества перед имеющимися аналогами.

Для определения сильных сторон разрабатываемой игры проведён анализ существующих реализаций игры. С учётом этого разработана спецификация требований к проектируемой системе.

1.1 Анализ существующих аналогов

По состоянию на март 2023 года на GitHub [1] по запросу «monopoly» доступно порядка 8 тысяч проектов, однако по запросу «monopoly client server» доступно всего лишь 15 проектов.

В таблице 1 приведены наиболее функциональные из них.

Таблица 1 — Критерии сравнения сервисов

Название	**Язык програм-мирования**	**Строк кода**	**Наличие сервера**	**Наличие системы комнат**	**Наличие рейтинговой игры**
monopoly	Java, JavaScript	11935	Да	Нет	Нет
CA314_monopoly	Java, Python	9877	Да	Нет	Нет
Tyranny	C++	17852	Да	Да	Нет

Исходя из данных о существующих клиент-серверных реализациях игры «Монополия» можно прийти к выводу, что ни одно из представленных приложений не реализует механизм рейтинговых игр, помимо этого не каждое решение поддерживает систему игровых комнат.

По результатам анализа существующих аналогов принято решение реализовать в разрабатываемой системе поддержку рейтинговых игр, игровых комнат и сохранения игрового процесса.

1.2 Описание особенностей реализации

В приложении А содержится полное описание основных правил и особенностей разрабатываемой реализации игры по мотивам «Монополии», с учётом выявленных ранее конкурентных преимуществ перед существующими аналогами с открытым исходным кодом

1.3 Разработка прецедентов

На рисунке 1 представлена диаграмма вариантов использования, отражающая действия клиента.

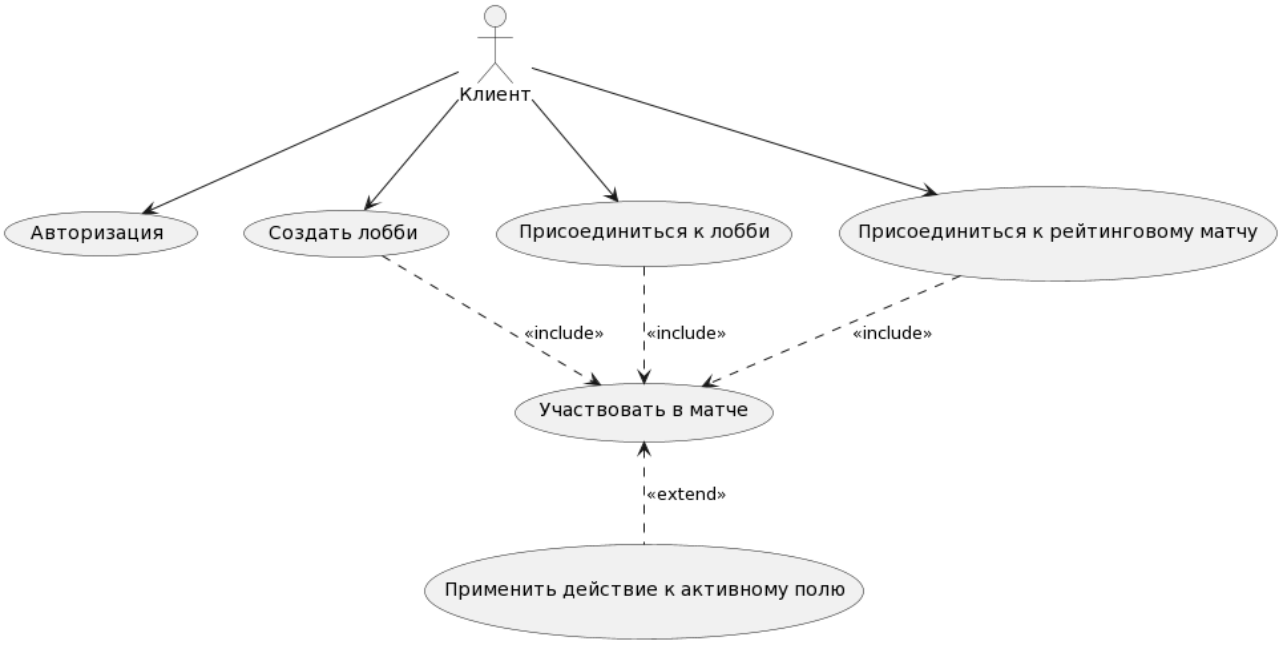

Рисунок 1 — Диаграмма вариантов использования в процессе игры

Название прецедента: Авторизация.

Предусловие: Клиент не авторизован.

Основная последовательность: Клиентское приложение запрашивает авторизацию на сервере. Получив от клиента авторизационную информацию

сервер выпускает на её основе авторизационные токены, и возвращает его клиенту.

Постусловие: Клиент становится авторизованным.

Название прецедента: Создать игровую комнату.

Предусловие: Клиент авторизован.

Основная последовательность: Клиентское приложение запрашивает создание игровой комнаты, сервер создаёт комнату и возвращает клиенту идентификатор созданной игровой комнаты.

Постусловие: Созданное игровой комнаты становится доступным для подключения других клиентов.

Условие для ввода в действие альтернативного сценария: Клиент не авторизован.

Альтернативная последовательность: Сервер сообщает клиенту, что он не авторизован.

Альтернативное постусловие: Игровая комната не создана.

Название прецедента: Присоединиться к игровой комнате.

Предусловие: Клиент авторизован, существует минимум одна комната.

Основная последовательность: Клиентское приложение запрашивает присоединение к комнате, сервер обрабатывает запрос клиента, по доступности этой операции возвращает клиенту детальную информацию о комнате.

Постусловие: Клиент присоединяется к комнате и ожидает начала игровой сессии.

Условие для ввода в действие альтернативного сценария 1: Клиент не авторизован.

Альтернативная последовательность 1: Сервер сообщает клиенту, что он не авторизован.

Альтернативное постусловие 1: Клиент не подключен к комнате.

Условие для ввода в действие альтернативного сценария 2: Клиентом передан неверный пароль.

Альтернативная последовательность 2: Сервер сообщает клиенту, что был передан неверный пароль.

Альтернативное постусловие 2: Клиент не подключен к комнате.

Условие для ввода в действие альтернативного сценария 3: Игровая комната заполнена.

Альтернативная последовательность 3: Сервер сообщает клиенту, что комната заполнена.

Альтернативное постусловие 3: Клиент не подключен к комнате.

Название прецедента: Присоединиться к рейтинговому матчу.

Предусловие: Клиент авторизован.

Основная последовательность: Клиентское приложение запрашивает присоединение к рейтинговому матчу, сервер обрабатывает запрос клиента, по доступности этой операции возвращает клиенту детальную информацию о рейтинговой игровой комнате.

Постусловие: Клиент присоединяется к рейтинговому матчу и ожидает начала игровой сессии.

Условие для ввода в действие альтернативного сценария: Клиент не авторизован.

Альтернативная последовательность: Сервер сообщает клиенту, что он не авторизован.

Альтернативное постусловие: Клиент не подключен к рейтинговому матчу.

Название прецедента: Участвовать в матче.

Предусловие: Клиент авторизован, игровая сессия в комнате запущена.

Основная последовательность: Сервером определяется порядок хода и цвета фишек игроков, эта информация передаётся клиентам. Клиент дожидается своего хода и в свой ход выбирает действие. Срабатывает условие окончания игры, определяется победитель, для рейтинговых игр начисляются очки рейтинга, игровая сессия завершается. Об этом сервер информирует клиентов.

Условие для ввода в действие альтернативного сценария: Клиент не авторизован.

Альтернативная последовательность: Сервер сообщает клиенту, что он не авторизован.

Альтернативное постусловие: Клиент не подключен к игровой сессии, но числится в ней.

1.4 Выводы по главе

Проведён анализ аналогов разрабатываемой системы, описаны требования и подробная спецификация к серверной системе. Приведённое описание диаграммы вариантов использования с разбором прецедентов позволит выполнить проектирование системы.

2 Проектирование

2.1 Определение сущностей разрабатываемой системы

На основании описания основных особенностей реализации, приведённых в разделе 1.1 становится понятно, что разрабатываемая система представляет собой сложное взаимодействие между различными её компонентами, и для успешного проектирования и разработки важно понимать их сущности и взаимосвязи.

– **пользователь (игрок)**: основная сущность, которая взаимодействует с игровой системой. Каждый пользователь имеет уникальный идентификатор, полученный от стороннего сервера авторизации. Этот идентификатор используется для определения уникальности игрока в системе и связывания его с игровым профилем. Игровой профиль хранит в себе имя игрока, его идентификатор и статистику;

– **игровая комната (lobby)**: специальная сущность, которая служит местом встречи для пользователей перед началом игровой сессии. Пользователи могут присоединяться к комнате, отсоединяться от неё или подготавливаться к началу игры. Каждый пользователь может находиться единовременно только в одной комнате. Владельцем игровой комнаты называется пользователь, создавший его или унаследовавший этот статус, он имеет возможность конфигурирования грядущей игровой сессии. Пользователя внутри комнаты и игровой сессии принято называть игроком;

– **игровая сессия**: сущность, представляющая собой конкретный экземпляр игрового процесса, созданный на основе конфигурации конкретной комнаты. Игровая сессия начинается, когда достигнуты все необходимые для этого условия внутри комнаты, например, все игроки подтвердили свою готовность, владелец запустил игру. В рамках игровой сессии, пользователи могут взаимодействовать с игровым интерфейсом — отражением сущности в

клиентском приложении, выполняя различные действия. По окончании сессии, результаты обрабатываются и сохраняются.

2.2 Выбор архитектуры

Среди множества многопользовательских игр довольно распространены проблемы с масштабированием и развитием серверного ПО. Во многом это связанно с типом выбранной архитектуры.

Как правило вышеобозначенными проблемами страдают приложения с монолитной архитектурой. Такой тип архитектуры достаточно эффективен на ранних стадиях эксплуатации системы, до тех пор, пока к ней не возникает новых требований.

В таблице 2 приводятся преимущества и недостатки монолитной архитектуры.

Таблица 2 — Преимущества и недостатки монолитной архитектуры

Преимущества	**Недостатки**
Легко организовать ведение системных журналов (логгирование), кэширование	Тяжело возвращаться к разработке, каждый раз происходит длительный процесс повторного изучения работы приложения
Легко первично развернуть приложение на целевой системе	Даже при незначительных изменениях в проекте требуется повторное развертывание всего приложения на целевой системе, что является относительно долгим процессом
Относительно несложно проводить сквозное тестирование	Возможности приложения ограничены выбором используемых технологий при старте проекта

Одной из альтернатив монолитной архитектуры является микросервисная архитектура. Концепция микросервисной архитектуры заключается в разбиении требований к системе на обособленные, слабо зависящие друг от друга компоненты. Микросервисная архитектура хорошо перекликается с философией UNIX — «Делай что-то одно, и делай это хорошо» [2].

В таблице 3 приводятся положительные и отрицательные аспекты микросервисной архитектуры.

Таблица 3 — Преимущества и недостатки микросервисной архитектуры

Преимущества	Недостатки
Компоненты слабо зависят друг от друга, отказ одного компонента не приводит к недоступности приложения в целом	Могут возникнуть сложности при сквозном тестировании целого приложения
Возможно масштабирование отдельных компонентов системы при достаточных ресурсах	Развертывание более трудозатратно, может потребоваться применение систем контейнеризации и оркестрации
Разработка приложения идёт быстрее из-за распределенной кодовой базы	Сложнее организовать мониторинг

Таким образом для реализации серверного приложения было принято решение остановить выбор архитектурного решения на микросервисах. В соответствии с принципами микросервисной архитектуры была разработана структурная схема архитектуры разрабатываемого решения, изображенная на рисунке 2.

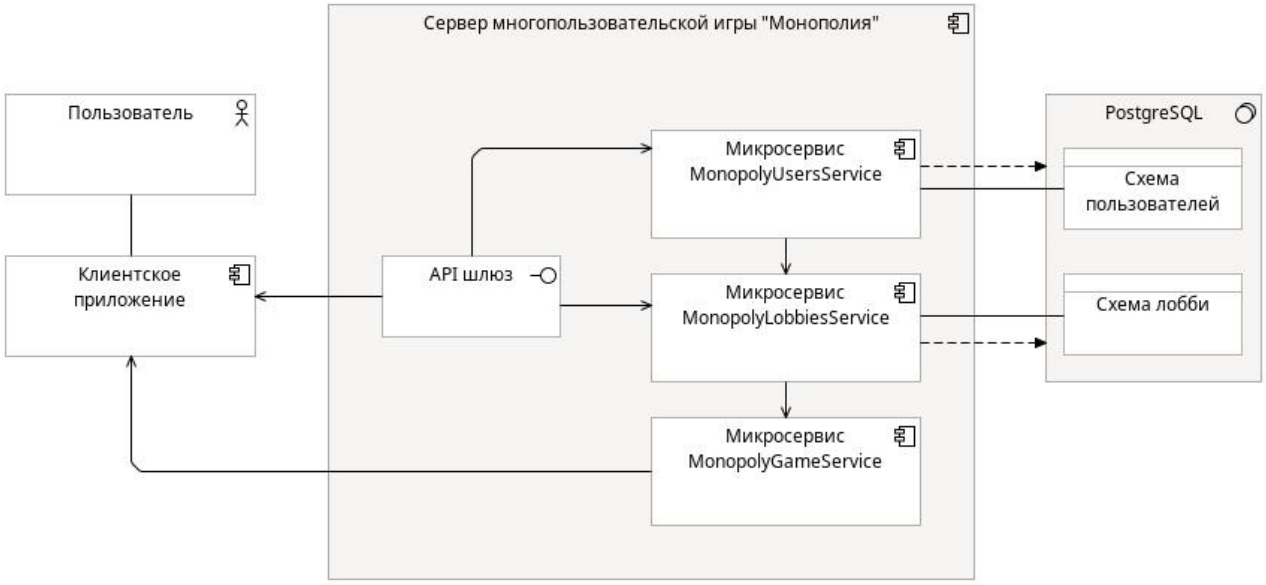

Рисунок 2 — Структурная схема архитектуры

Предполагается отдельная реализация HTTP API для внеигровых процессов в виде микросервиса «API шлюз». Игровые процессы предполагается обрабатывать на уровне взаимодействия сетевых сокетов с пользованием протокола сериализации Protocol Buffers (ProtoBuf, PB) [3] в микросервисе «MonopolyGameService». Для обработки действий с сущностями пользователей

и игровых комнат предполагается использовать микросервисы «MonopolyUsersService» и «MonopolyLobbiesService», соответственно.

Общение между микросервисами планируется осуществлять с помощью протокола удаленного вызова процедур gRPC [4].

2.3 Проектирование микросервиса «API шлюз»

Микросервис «API шлюз» необходим для коммутирования пользовательских запросов в нужный микросервис для внутренней обработки. Для каждой функциональности микросервисов «MonopolyUsersService» и «MonopolyLobbiesService» должен быть предусмотрен соответствующий HTTP метод. Подробное описание внутренней логики каждого метода приводится в разделах 2.4.2 и 2.5.1, соответственно.

2.3.1 Разработка HTTP методов

В таблице 4 приведён список HTTP методов проектируемого микросервиса с кратким описанием и указанием микросервиса, реализующего этот метод. Расширенное описание каждого HTTP метода приводится под таблицей.

Таблица 4 — HTTP методы

№	Имя	Тип	Описание	Сервис
1	/auth/authFromVK	POST	Авторизация с помощью учётной записи «VK»	Users
2	/auth/authFromGoogle	POST	Авторизация с помощью учётной записи «Google»	Users
3	/auth/authAsGuest	POST	Гостевая авторизация	Users
4	/auth/refreshAccessToken	POST	Перевыпуск токена доступа	Users
5	/users/me/changeNickname	POST	Изменение игрового имени	Users
6	/users/{id}/getInfo	GET	Получение информации о пользователе с id={id}	Users
7	/lobbies/create	POST	Создание комнаты	Lobbies
8	/lobbies/getList	GET	Получение списка комнат	Lobbies
9	/lobbies/{id}/connect	POST	Подключение к комнате с id={id}	Lobbies

Окончание таблицы 4

№	Имя	Тип	Описание	Сервис
10	/lobbies/ranked/connect	POST	Подключение к рейтинговой комнате	Lobbies
11	/lobbies/active/check	GET	Проверка активной игровой сессии	Lobbies
12	/lobbies/current/getInfo	GET	Получение информации о текущей комнате	Lobbies
13	/lobbies/current/ updateSettings	POST	Обновление настроек текущей комнаты	Lobbies
14	/lobbies/current/ switchReadiness	POST	Изменение состояния готовности	Lobbies
15	/lobbies/current/players/{id}/ raise	POST	Передача прав управления комнатой игроку с id={id}	Lobbies
16	/lobbies/current/players/{id}/ kick	POST	Исключение из текущей комнаты игрока с id={id}	Lobbies
17	/lobbies/current/disconnect	POST	Отключение от комнаты	Lobbies
18	/lobbies/current/run	POST	Запуск комнаты	Lobbies
19	/lobbies/{id}	DEL	Удаление комнаты с id={id}	Lobbies

Запросы ко всем методам, кроме авторизационных должны содержать заголовок *Authorization: bearer <token>*.

Для запросов к методу /auth/refreshAccessToken *<token>* — токен перевыпуска, для запросов к остальным методам *<token>* — токен доступа. На основе этого токена микросервис идентифицирует пользователя и разрешает/запрещает доступ к методу.

Если ответом на не авторизационный запрос получен ответ HTTP 401 Unauthorized, то клиентскому приложению необходимо перевыпустить токен доступа используя метод POST /auth/refreshAccessToken. Если на запрос к этому методу также получен ответ HTTP 401 Unauthorized, то клиентскому приложению необходимо повторно запросить авторизацию у пользователя. Подробное описание процесса авторизации приводится в разделе 2.4.1.

Тело любого HTTP 200 OK ответа от авторизационного метода будет содержать JSON структуру «tokens». В приложении Б содержатся примеры всех массивных JSON сруктур.

Для любого метода, предусматривающего его выполнение владельцем комнаты, на запрос не от владельца ответом будет служить HTTP 403 Forbidden.

Ниже приводится описание HTTP методов:

2.3.1.1 **метод POST /users/authFromVK:** Данный метод реализует авторизацию пользователя с использованием стандарта OAuth2.0 [5] в реализации компании «VK» [6]. Тело запроса должно соответствовать следующему образцу в формате JSON:

{"code": "<code>"}

Где <*code*> — код, полученный клиентским приложением в результате авторизации в сервисах VK.

2.3.1.2 **метод POST /users/authFromGoogle:** Данный метод реализует авторизацию пользователя с использованием стандарта OAuth2.0 в реализации компании «Google» [7]. Тело запроса должно быть таким же как и для метода /users/authFromVK, где <*code*> — код, полученный клиентским приложением в результате авторизации в сервисах Google.

2.3.1.3 **метод POST /users/authAsGuest:** Данный метод реализует авторизацию пользователя без привязки учётной записи (гостевую авторизацию). Тело запроса должно быть таким же как и для метода /users/authFromVK, где <*code*> — уникальный сгенерированный клиентским приложением ключ.

2.3.1.4 **метод POST /auth/refreshAccessToken:** Данный метод реализует перевыпуск токена доступа пользователя. В заголовке запроса заголовок *Authorization* должен содержать токен перевыпуска. Тело запроса не требуется.

2.3.1.5 **метод POST /users/me/changeNickname:** Данный метод реализует изменение игрового имени пользователя. Тело запроса должно соответствовать следующему образцу в формате JSON:

{"newNickname": "Bob"}

2.3.1.6 **метод GET /users/{id}/getInfo:** Данный метод реализует получение информации о пользователе. Тело запроса не требуется. Тело HTTP 200 OK ответа на запрос будет содержать JSON структуру «userInfo», изображённую на рисунке 3, на следующей странице.

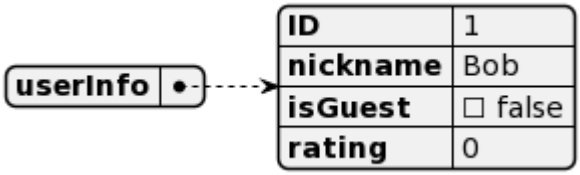

Рисунок 3 — Пример JSON структуры «userInfo»

2.3.1.7 **метод POST /lobbies/create:** Данный метод реализует создание комнаты. Тело запроса может содержать JSON структуру «settings», изображённую на рисунке 4:

Рисунок 4 — Пример JSON структуры «settings» при создании комнаты

Значение поля «*type*» может быть только «*PRIVATE*» или «*PUBLIC*», во втором случае поле «*password*» указывать не обязательно. Значение поля «*victoryType*» может быть только «*SCORE*», «*TURN*» или «*BOTH*», поле «*scoreVictoryValue*» или «*turnVictoryValue*» может быть опущено за ненадобностью.

Тело HTTP 200 OK ответа будет содержать JSON структуру «lobbyInfo», изображенную на рисунке 5, на следующей странице.

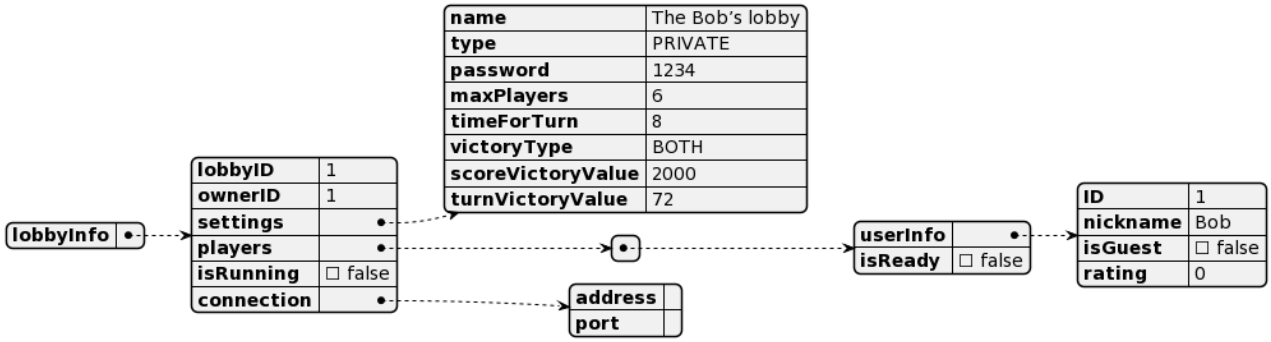

Рисунок 5 — Вид JSON структуры «lobbyInfo» при создании комнаты игроком Bob

2.3.1.8 **метод GET /lobbies/getList:** Данный метод реализует получение списка не рейтинговых комнат. Тело запроса не требуется. Тело HTTP 200 OK ответа будет содержать JSON структуру «lobbies», изображенную на рисунке 6:

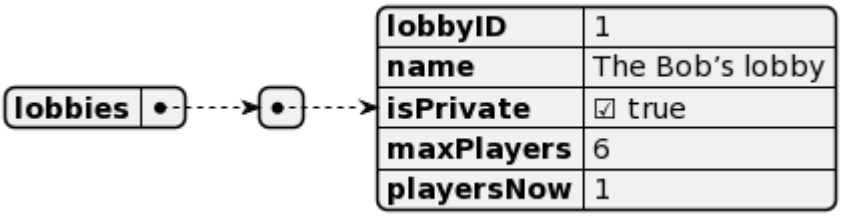

Рисунок 6 — Пример JSON структуры «lobbies»

2.3.1.9 **метод POST /lobbies/{id}/connect:** Данный метод реализует подключение игрока к не рейтинговой комнате. Тело запроса может соответствовать следующему образцу в формате JSON:

{"password": "1234"}

Если комната публичная, то поле «*password*» является опциональным, в таком случае необходимо передавать пустую JSON структуру. В случае несоответствия (или отсутствия) пароля при подключении к приватной комнате сервером будет возвращен HTTP 403 Forbidden.

Тело HTTP 200 OK ответа будет содержать JSON структуру «lobbyInfo», изображенную на рисунке 7, на следующей странице.

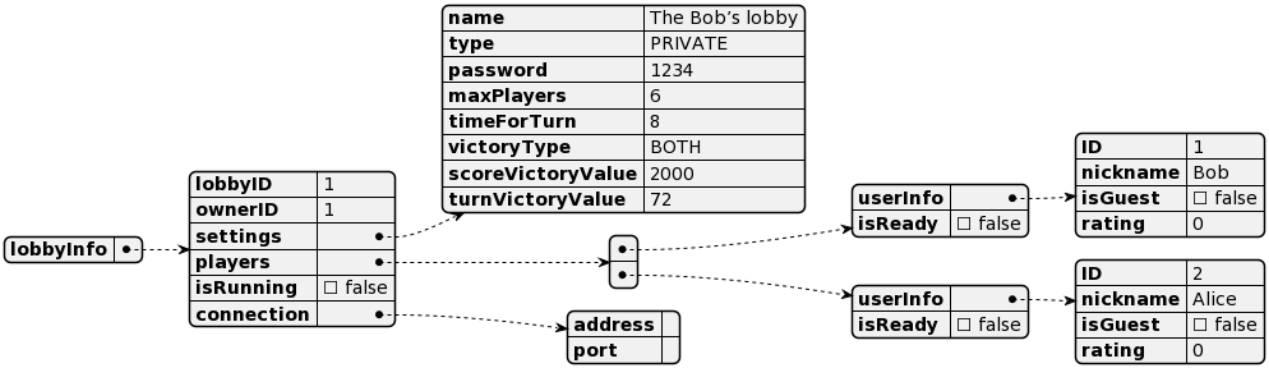

Рисунок 7 — Вид JSON структуры «lobbyInfo» после подключения игрока Alice к комнате игрока Bob

2.3.1.10 **метод POST /lobbies/ranked/connect:** Данный метод реализует подключение игрока к не рейтинговой комнате. Тело запроса не требуется. Тело HTTP 200 OK ответа будет содержать JSON структуру «lobbyInfo» рейтинговой комнаты.

2.3.1.11 **метод GET /lobbies/active/check:** Данный метод реализует проверку активных игровых сессий с участием пользователя. Тело запроса не требуется.

В случае если у пользователя есть активная игровая сессия, то ответом будет возвращён HTTP 200 OK, а тело будет содержать JSON структуру «connection», изображённую на рисунке 8, иначе будет возвращён ответ HTTP 404 Not Found.

Рисунок 8 — Пример JSON структуры «connection»

2.3.1.12 **метод GET /lobbies/current/getInfo:** Данный метод реализует обновление информации о текущей комнате. Тело запроса не требуется. Тело HTTP 200 OK ответа будет содержать актуальную JSON структуру «lobbyInfo».

2.3.1.13 **метод POST /lobbies/current/updateSettings:** Данный метод реализует обновление настроек текущей комнаты её владельцем. Тело запроса должно содержать JSON структуру «settings». Тело HTTP 200 OK ответа будет содержать обновлённую JSON структуру «lobbyInfo».

2.3.1.14 **метод POST /lobbies/current/switchReadiness:** Данный метод реализует изменение состояния готовности игрока внутри комнаты. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK.

2.3.1.15 **метод POST /lobbies/current/players/{id}/raise:** Данный метод реализует передачу владельцем прав управления комантой другому игроку. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK, а пользователь перестанет быть владельцем комнаты.

2.3.1.16 **метод POST /lobbies/current/players/{id}/kick:** Данный метод реализует исключение владельцем другого игрока из комнаты. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK.

2.3.1.17 **метод POST /lobbies/current/disconnect:** Данный метод реализует отключение игрока от комнаты. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK.

2.3.1.18 **метод POST /lobbies/current/run:** Данный метод реализует запуск игровой сессии владельцем комнаты вне зависимости от состояния готовности игроков и их количества. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK.

2.3.1.19 **метод DELETE /lobbies/{id}:** Данный метод реализует удаление комнаты её владельцем. Тело запроса не требуется. В случае успеха сервером будет возвращен ответ HTTP 200 OK, а комната будет удалена. Все игроки, находившиеся в комнате на свой GET запрос /lobbies/current/getInfo получат ответ HTTP 404 Not Found.

2.3.2 Диаграмма последовательности

На рисунке 9 приведена диаграмма последовательности для проектируемого микросервиса. Входящие авторизованные HTTP запросы перенаправляются в целевой микросервис для выполнения логики запроса.

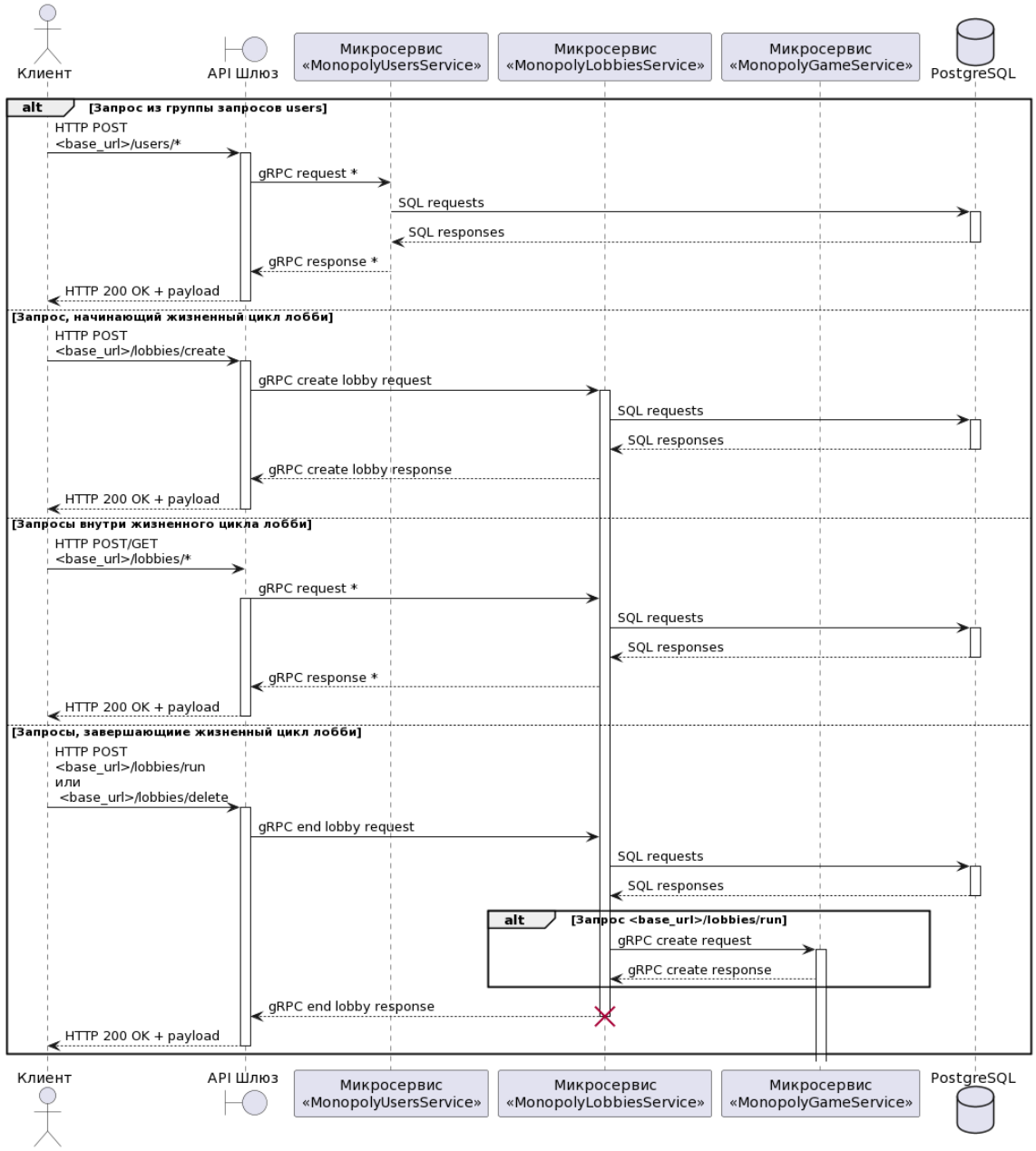

Рисунок 9 — Диаграмма последовательности для микросервиса «API шлюз»

2.4 Проектирование микросервиса «MonopolyUsersService»

Микросервис «MonopolyUsersService» служит для обработки пользовательских запросов, пропущенных микросервисом «API шлюз», и служебных запросов со стороны остальных микросервисов. Микросервис реализует операции над сущностями типа «Пользователь».

2.4.1 Процесс авторизации

Помимо прямых операций над сущностями типа «Пользователь» микросервис выполняет функцию сервера авторизации пользователей и их регистрацию в системе. Авторизация пользователей в разрабатываемой системе как и упоминалось ранее выполнена на основе JSON Web Tokens (JWT) [8].

На основе данных, полученных от пользователя или стороннего сервера авторизации микросервис идентифицирует пользователя и генерирует два JWT с различными сроками действия, и предназначенных для разных действий:

– access_token: токен доступа, служащий для проеверки авторизации HTTP запросов микросервисом «API шлюз». Такой токен имеет недлительное время действия;

– refresh_token: токен перевыпуска, служащий для выпуска нового access_token.

Для подписи токена используется симметричный алгоритм HS256 [9] и секретный ключ с достаточной энтропией, известный только микросервисам «MonopolyUsersService» и «API шлюз». В качестве полезной нагрузки внутрь JWT помещается:

– тип токена: refresh или access;

– идентификатор пользователя;

– временная метка истечения срока действия токена.

Такой подход позволяет микросервису «API шлюз» по HTTP заголовку Authorization идентифицировать авторизированного пользователя без надобности обращения к базе данных, и сразу же инициировать обработку пользовательского запроса в нужном микросервисе.

На рисунке 10 приведена диаграмма последовательности авторизации с использованием сторонней реализации стандарта OAuth2.0:

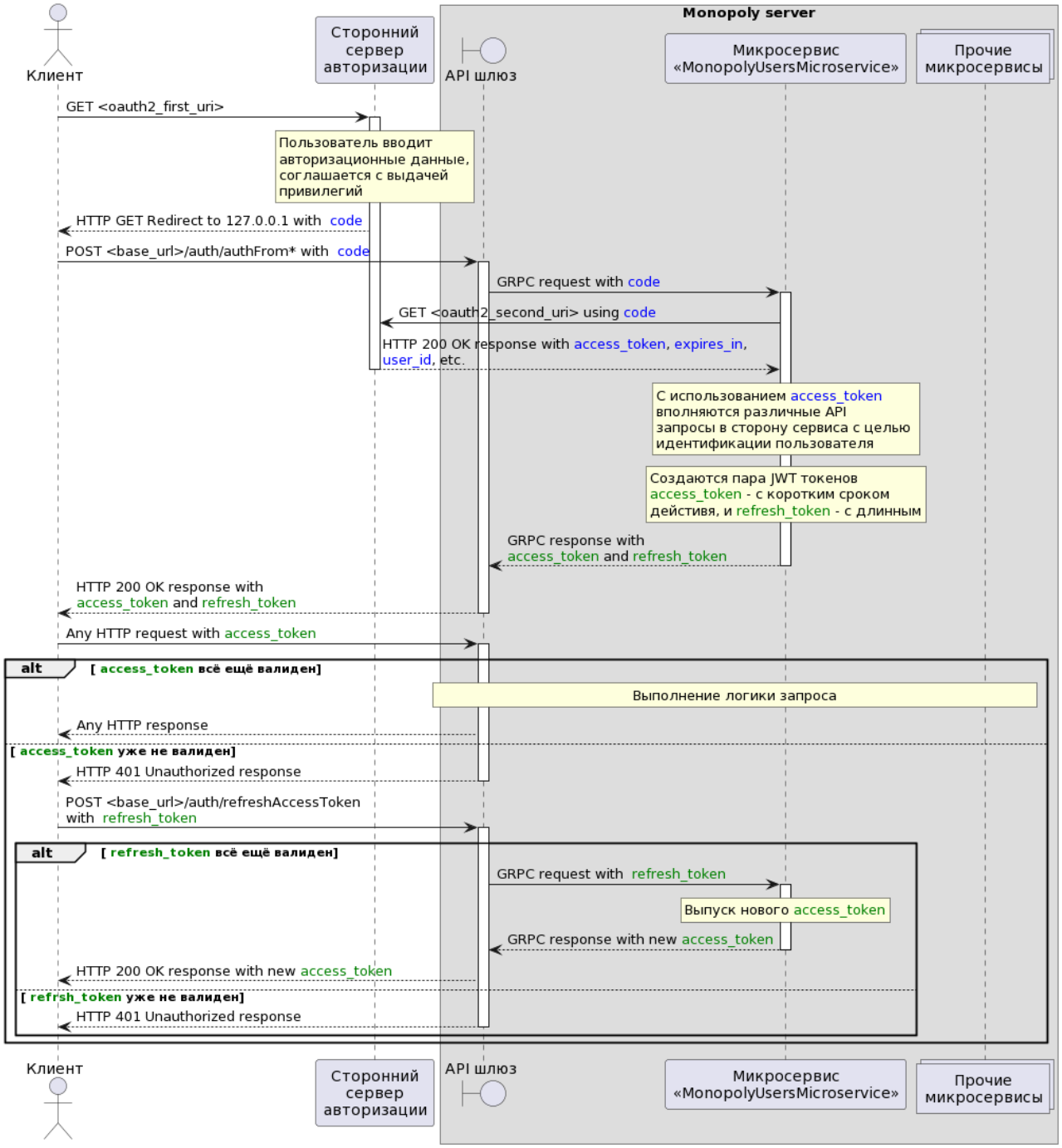

Рисунок 10 — Диаграмма последовательности OAuth2.0 авторизации

2.4.2 Описание удалённых процедур

Проектируемый микросервис должен представлять из себя gRPC сервер, поддерживающий Remote Procedure Calls (RPC) приведённых в таблице 5 удалённых процедур, описание приводится под таблицей.

Таблица 5 — Удалённые процедуры микросервиса «MonopolyUsersService»

№	Имя	Принимаемая структура	Возвращаемая структура
1	AuthFromVK	AuthRequest	AuthResponse
2	AuthFromGoogle	AuthRequest	AuthResponse
3	AuthAsGuest	AuthRequest	AuthResponse
4	RefreshAccessToken	RefreshAccessTokenRequest	RefreshAccessTokenResponse
5	GetInfo	GetInfoRequest	GetInfoRequest
6	ChangeNickname	ChangeNicknameRequest	ChangeNicknameRequest

Описание PB структур проектируемого микросервиса приводится в приложении В. Для всех успешных RPC поле «status» в структуре ответа будет равно 0 — «SUCCESS».

Ниже приводится описание удалённого вызова каждой процедуры:

2.4.2.1 **RPC AuthFromVK:** процедурой реализуется формирование пары JWT согласно схеме, описанной в разделе 2.4.1, в качестве стороннего сервера авторизации используется сервер авторизации компании «VK». В случае ошибки, возвращенной внешним сервером авторизации поле «status» в структуре ответа будет равно 3 — «FAILED_DEPENDENCY».

2.4.2.2 **RPC AuthFromGoogle:** процедурой реализуется формирование пары JWT согласно схеме, описанной в разделе 2.4.1, в качестве стороннего сервера авторизации используется сервер авторизации компании «Google». В случае ошибки, возвращенной внешним сервером авторизации поле «status» в структуре ответа будет равно 3 — «FAILED_DEPENDENCY».

2.4.2.3 **RPC AuthAsGuest:** процедурой реализуется формирование пары JWT согласно схеме, описанной в разделе 2.4.1, за исключением взаимодействия со сторонним сервером авторизации, на основании уникального идентификатора

клиентского приложения. Пользователь будет зарегистрирован с пометкой «isGuest», для такого пользователя не собирается игровая статистика и не доступны рейтинговые игры. После истечения refresh_token учётная запись такого пользователя удаляется.

2.4.2.4 **RPC RefreshAccessToken:** процедурой реализуется формирование нового access_token согласно схеме, описанной в разделе 2.4.1.

2.4.2.5 **RPC GetInfo:** процедурой реализуется возвращение информации о пользователе с идентификатором, указанным в поле «targetUser» структуры запроса.

2.4.2.6 **RPC ChangeNickname:** процедурой реализуется изменение игрового имени пользователя на содержащиеся в поле «newNickname» структуры запроса имя.

2.5 Проектирование микросервиса «MonopolyLobbiesService»

Микросервис «MonopolyLobbiesService» служит для обработки пользовательских и служебных запросов, связанных с сущностями типа «Игровая комната».

2.5.1 Описание удалённых процедур

Проектируемый микросервис должен представлять из себя gRPC сервер, поддерживающий RPC приведённых в таблице 6 удалённых процедур, описание приводится под таблицей.

Таблица 6 — Удалённые процедуры микросервиса «MonopolyUsersService»

№	Имя	Принимаемая структура	Возвращаемая структура
1	Create	SetSettingsRequest	AboutLobbyResponse
2	GetList	RequesterOnlyRequest	GetListResponse
3	CheckInActive	RequesterOnlyRequest	CheckInActiveResponse
4	Connect	ConnectRequest	AboutLobbyResponse
5	ConnectToRanked	RequesterOnlyRequest	AboutLobbyResponse

Окончание таблицы 6

№	Имя	Принимаемая структура	Возвращаемая структура
6	GetInfo	RequesterOnlyRequest	AboutLobbyResponse
7	UpdateSettings	SetSettingsRequest	StatusOnlyResponse
8	SwitchReadiness	RequesterOnlyRequest	StatusOnlyResponse
9	RaisePlayer	RequesterAndTargetRequest	StatusOnlyResponse
10	KickPlayer	RequesterAndTargetRequest	StatusOnlyResponse
11	Disconnect	RequesterOnlyRequest	StatusOnlyResponse
12	Run	RequesterOnlyRequest	StatusOnlyResponse
13	Delete	RequesterOnlyRequest	RequesterOnlyRequest
14	RegisterPlayerConnection	RequesterOnlyRequest	StatusOnlyResponse
15	RegisterPlayerDisconnection	RequesterOnlyRequest	StatusOnlyResponse
16	RegisterGameResults	RegisterResultsRequest	StatusOnlyResponse

Описание PB структур проектируемого микросервиса приводится в приложении Г. Для всех успешных RPC поле «status» в структуре ответа будет равно 0 — «SUCCESS». Во всех случаях, когда запрашивается информация или действие, предполагающее принадлежность пользователя к игровой комнате, но без выполнения этого условия поле «status» в структуре ответа будет равно 4 — «RESOURCE_NOT_AVAILABLE». В случае, если пользователем, не являющимся владельцем комнаты, запрашивается действие, требующие привилегии владельца — поле «status» в структуре ответа будет равно 7 — «FORBIDDEN». Пользователь определяется на основании соответсвия его идентификатора с идентификатором, указанным в поле «requesterID» структуры запроса.

Важно понимать, что одна сущность типа «Пользователь» единовременно может быть связанна только с одной сущностью типа «Игровая комната», в силу этого при возникновении конфликта при создании или присоединении к комнате, если игрок является участником другой команты, то он будет отключен от неё.

Ниже приводится описание удалённого вызова каждой процедуры:

2.5.1.1 **RPC Create:** процедурой реализуется создание комнаты на основе предоставленных пользователем настроек, если настройки не предоставлены, то комната создаётся на основе заранее предустановленных настроек, если настройки содержат недопустимые данные, то в структуре ответа поле «status»

будет равно 6 — «BAD_VALUE». После создания комнаты игрок, создавший её будет назначен её владельцем и автоматически к ней подключён.

2.5.1.2 **RPC GetList:** процедурой реализуется возвращение списка краткой информации о каждой не запущенной не рейтинговой комнате.

2.5.1.3 **RPC CheckInActive:** процедурой реализуется проверка присутствия пользователя в любом запущенной игровой сессии, кроме рейтинговой. Если результат проверки положительный, то в структуре ответа в поле «connection» будут содержаться данные для подключения к активной игровой сессии.

2.5.1.4 **RPC Connect:** процедурой реализуется подключение игрока к комнате с идентификатором, указанным в поле «lobbyID». Опциональное поле «password» проверяется в случае, если комната является приватной.

2.5.1.5 **RPC ConnectToRanked:** процедурой реализуется подключение игрока к уже созданной рейтинговой комнате, либо создание рейтинговой комнаты от имени этого игрока и подключения к ней. Стоит отметить, что хоть пользователь в таком случае и будет считаться владельцем рейтинговой комнаты, то никаких привилегий владельца ему это не даёт.

2.5.1.6 **RPC GetInfo:** процедурой реализуется предоставление пользователю полной информации о его текущей комнате.

2.5.1.7 **RPC UpdateSettings:** процедурой реализуется обновление текущих настроек комнаты её владельцем. Обработка настроек реализуется тем же механизмом, что и в RPC Create.

2.5.1.8 **RPC SwitchReadiness:** процедурой реализуется переключение состояния готовности игрока внутри его комнаты. Если комната заполнена игроками, и каждый игрок подтвердил свою готовность, то микросервисном инициализируется игровая сессия.

2.5.1.9 **RPC RaisePlayer:** процедурой реализуется передача владельцем комнаты привилегий владельца игроку с идентификатором, указанным в поле «targetID» структуры запроса. Если игрока с таким идентификатором в игровой

комнате нет, то в структуре ответа поле «status» будет равно 4 — «RESOURCE_NOT_AVAILABLE».

2.5.1.10 **RPC KickPlayer:** процедурой реализуется исключение из комнаты игрока с идентификатором, указанным в поле «targetID» структуры запроса. Если игрока с таким идентификатором в комнате нет, то в структуре ответа поле «status» будет равно 4 — «RESOURCE_NOT_AVAILABLE».

2.5.1.11 **RPC Disconnect:** процедурой реализуется отключение игрока от комнты. Если отключающийся игрок является владельцем комнаты, то среди оставшихся игроков определяется новый владелец. Если игроков в комнате больше не остаётся, то она принудительно удаляется.

2.5.1.12 **RPC Run:** процедурой реализуется принудительный запуск игровой сессии владельцем комнаты, происходит инициализация игровой сессии.

2.5.1.13 **RPC Delete:** процедурой реализуется удаление комнаты, каждый игрок принудительно отсоединяется от неё.

2.5.1.14 **RPC RegisterPlayerConnection:** процедурой реализуется изменение внутреннего состояния игрока внутри комнаты на состояние, соответствующее присоединению игрока к игровой сессии.

2.5.1.15 **RPC RegisterPlayerDisconnection:** процедурой реализуется изменение внутреннего состояния игрока внутри комнаты в структуре базы данных на состояние, соответствующее отсоединению игрока от игровой сессии.

2.5.1.16 **RPC RegisterGameResults:** процедурой реализуется регистрация результатов игровой сессии и завершение жизненного цикла игровой комнаты.

2.5.2 Инициализация игровой сессии

Игровая сессия инициализируется в результате выполнения RPC Run владельцем комнаты, либо в случае выполнения всех стандартных условий. При инициализации проектируемый микросервис обращается к микросервису

«MonopolyGameService», получает от него данные для подключения к игровой сессии и обновляет состояние комнаты «isRunning» на True. Наличие этого флага и структуры «connection» в структуре «AboutLobbyResponse», возвращаемой некоторыми RPC, и впоследствии преобразованной в JSON и доставленной клиентскому приложению информирует его о запуске игровой сессии.

Подробное описание инициализации игровой сессии на стороне микросервиса «MonopolyGameService» приведено в разделе 2.6.

2.6 Проектирование микросервиса «MonopolyGameService»

Микросервис «MonopolyGameService» предназначается для обработки сущностей типа «Игровая сессия». Для связи проектируемого микросервиса с остальными микросервисами разрабатываемой системы предусматривается реализация gRPC сервера. Сетевое взаимодействие с клиентами осуществляется на уровне взаимодействия TCP сокетов. Выбор транспортного протокола обусловлен пошаговой составляющей игрового процесса — важна гарантия и порядок доставки сообщений и относительно не важна скорость их доставки.

gRPC сервером реализуется возможность удалённого вызова процедуры Run служащей для инициализации игровой сессии на уровне проектируемого микросервиса на основании полной информации о комнате на момент запуска. Для игровой сессии выделяется программный порт, который в сочетании с сетевым адресом образует необходимую для подключения клиентов структуру «Connection», которая в конечном итоге возвращается клиенту. Выделенный на игровую сессию порт является её идентификатором внутри «MonopolyGameService».

Для общения между клиентом и сервером используются несколько основных сообщений:

– сообщение ClientHello: первое сообщение в протоколе, отправляемое клиентом после установления соединения с сервером. Получив такое сообщение

сервер проверяет содержится ли переданный идентификатор пользователя в проинициализированной микросервисом «MonopolyLobbiesService» информации об игровой сессии;

– **сообщение ServerHello:** второе сообщение в протокле, отправляемое сервером в подтверждение существования игрока-клиента в игровой сессии, содержащее первичную информацию, необходимую клиенту. В противном случае сервер разорвёт соеденение;

– **сообщение StateRefresh:** основное сообщение с указанием сервером общего состояния игровой сессии для каждого игрока, и ожидаемого от клиента действия;

– **сообщения-сигналы:** ряд сообщений, отправляемых клиентом и информирующих сервер о выполннении клиентским приложением соответствующих служебных действий (отображение перемещения фишки игрока, броска игровых костей и т.д.). Тип такого сообщения передаётся сервером в составе сообщения StateRefresh;

– **сообщения-решения:** ряд сообщений, отправляемых клиентом по результату выполнения пользователем каких-либо действий (покупка актива, пропуск хода и т.д.). Тип такого сообщения и целевая клетка передаются сервером в составе сообщения StateRefresh;

Сообщения между клиентом и сервером сериализуются с помощью Protocol Buffers. Разработанный протокол клиент-серверного взаимодействия и Proto файл gRPC сервера приводятся в приложении Д.

На рисунке 11, на следующей странице изображена диаграмма последовательности, отражающая порядок течения игровой сессии и клиент-серверное взаимодействие.

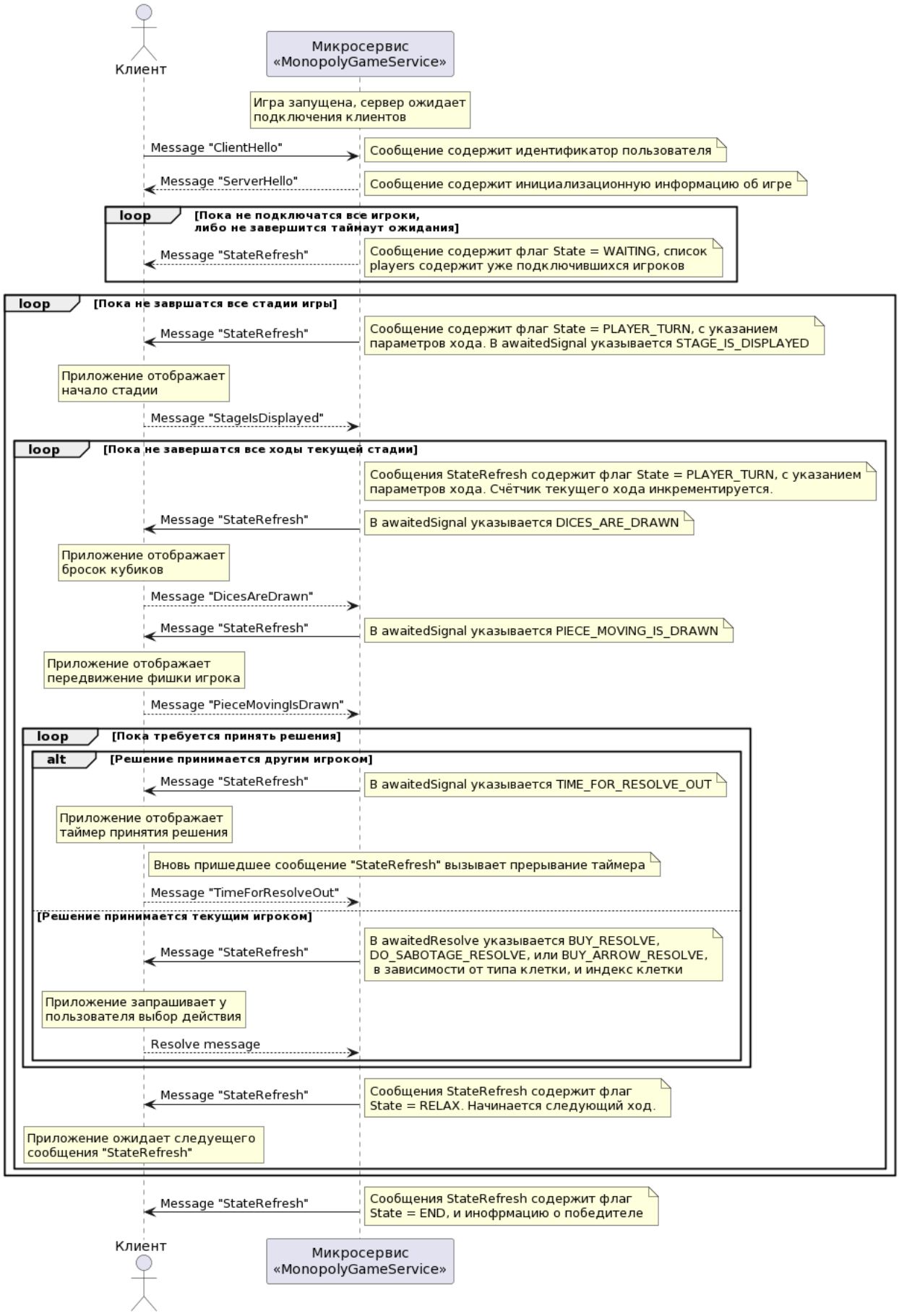

Рисунок 11 — Диаграмма последовательности игровой сессии

2.7 Проектирование структуры базы данных

Основываясь на анализе сущностей в предметной области и проектировании микросервисов становится возможным определить структуру базы данных. Так как существуют явные взаимосвязи между сущностями, то структура базы данных должна быть выполнена в соответствии с реляционной теорией. Взаимодействие сущностей базы данных в виде ER-диаграммы изображено на рисунке 12:

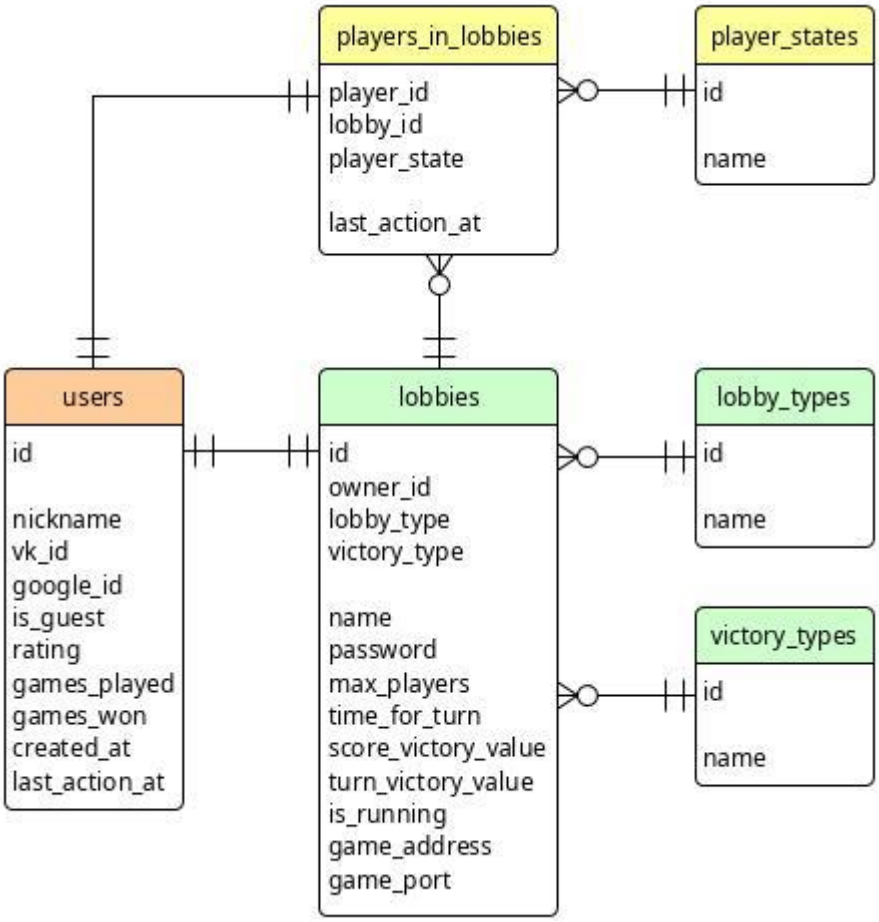

Рисунок 12 — ER-диаграмма структуры базы данных

Основные связи на этой диаграмме: каждый конкретный пользователь единовременно может быть владельцем только одной комнаты, а внутри каждой комнаты может быть несколько игроков.

2.8 Выводы по главе

Осуществлён выбор архитектуры разрабатываемой системы, выполнено проектирование микросервисов серверного приложения многопользовательской игры «Монополия». Определены протоколы межкомпонентного взаимодействия и клиент-серверного взаимодействия в процессе игровой сессии.

На основании этого определяются подходы к реализации, тестированию и развертыванию разрабатываемой серверной системы, описанные в следующей главе.

3 Реализация и тестирование

3.1 Разработка

На протяжении всего цикла разработки использовалась система контроля версий git [10]. В качестве удаленного хранилища для репозиториев выбран онлайн сервис GitHub.

3.1.1 Разработка микросервисов «MonopolyGameService» и «API шлюз»

Микросервисы «MonopolyGameService» и «API шлюз», являясь наиболее нагруженными, разрабатывались на высокоуровневом языке программирования Golang [11] с использованием его следующих программных модулей:

– flag: используется для чтения аргументов командной строки;

– ioutil: используется для функций ввода-вывода;

– toml: используется для обработки конфигурационных файлов формата .toml;

– log: используется для ведения системных журналов;

– os: используется для взаимодействия с ресурсами операционной системы;

– net/http: используется для реализации HTTP сервера;

– crypto/tls: используется для реализации безопасного соединения;

– strings: используется для работы со строковыми переменными;

– strvconv: используется для конвертации нестроковых типов данных в строки;

– jwt: используется «API шлюзом» для верификации JWT;

– mux: используется «API шлюзом» для мультиплексирования HTTP запросов по функциям-обработчикам;

– context: используется для работы с контекстами;

– errors: используется для работы с ошибками;

– protojson: используется «API шлюзом» для конвертирования Proto структур в формат JSON;

– encoding/json: используется для заполнения структур на основе JSON;

– grpc: используется для реализации gRPC клиента (для «API шлюза») и сервера (для «MonopolyGameService»).

Исходные коды обоих микросервисов оформлены в виде нескольких отдельных модулей с общими точками входа.

Модули микросервиса «API шлюз»:

– mp_config: для работы с конфигурацией микросервиса;

– mp_http: для обработки входящих HTTP запросов и формирования ответов;

– mp_grpc: для вызова удалённых процедур остальных микросервисов. Реализована работа HTTP сервера как в небезопасном (HTTP), так и в безопасном (HTTPS) режимах.

Модули микросервиса «API шлюз»:

– mp_config: для работы с конфигурацией микросервиса;

– mp_core: для реализации ядра микросервиса;

– mp_game: для обработки игровой сессии;

– mp_grpc: для реализации gRPC сервера;

– mp_net: для сетевого взаимодействия с игровыми клиентами.

Конкурентность обоих программ достигается за счёт легковесных потоков, предоставляемых языком Golang — горутин (goroutines).

Git-репозиторий с исходным кодом микросервиса «API шлюз» размещён на GitHub [12]. Git-репозиторий с исходным кодом микросервиса «MonopolyGameService» так же размещён на GitHub [13].

3.1.2 Разработка микросервисов «MonopolyUsersService» и «MonopolyLobbiesService»

Микросервисы «MonopolyUsersService» и «MonopolyLobbiesService» схожи и по своей внутренней структуре, и по характеру решаемых задач — обработка действия пользователей вне игровой сессии, поэтому было принято решение разрабатывать их на основе одних и тех же инструментов. Таким образом разработка велась на высокоуровневом языке программирования Python3 с использованием его следующих программных пакетов:

– sys: используется для взаимодействия с интерпретатором языка;

– logging: используется для ведения системных журналов;

– grpc: используется для реализации gRPC сервера;

– time, datetime: используются для работы со временем;

– json: используется для работы с JSON структурами;

– aiohttp: используется «MonopolyUsersService» для внешних HTTP запросов;

– vk_api: используется «MonopolyUsersService» для получения имени пользователя;

– jwt: используется «MonopolyUsersService» для выпуска JWT;

– asyncio: используется для общей асинхронности;

– psycopg2: используется для запросов к базе данных PostgreSQL.

Оба микросервиса реализуют gRPC сервер, для этого в каждом реализован класс Listener, методами которых реализуется внутренняя логика запроса. Для общения с базой данных PostgrSQL в обоих микросервисах реализован класс DatabaseManager в составе модуля database.

Git-репозиторий с исходным кодом микросервиса «MonopolyUsersService» размещён на GitHub [14]. Git-репозиторий с исходным кодом микросервиса «MonopolyUsersService» так же размещён на GitHub [15].

3.2 Тестирование

В соответствии с требованиями к разрабатываемой системе, на протяжении всей стадии разработки происходило систематическое ручное тестирование (manual testing). В рамках процесса тестирования находились и исправлялись дефекты, ниже приведены некоторые из них:

– допущена возможность исключение самого себя с использованием HTTP метода /lobbies/current/players/{id}/kick: дефект устранён;

– допущена возможность исключения игрока из другой игровой комнаты: дефект устранён;

– упущена валидация входных параметров при обновлении настроек игровой комнаты: дефект устранён;

– допущено отображение запущенных комнат: дефект устранён.

Помимо ручного тестирования проводилось нагрузочное тестирование API, показавшее приемлемый результат.

3.3 Развёртывание

Развёртывание каждого микросервиса возможно с использованием системы контейнеризации Docker [16]. Для этого в репозиториях подготовлены сборочные Dockerfile. Также возможен выпуск RPM-пакетов для установки на операционные системы семейства RedHat Linux.

Помимо самих микросервисов с помощью Docker возможно развёртывание базы данных с подготовленной схемой.

Помимо развёртывания так же важна правильная конфигурация каждого микросервиса для корректной совместной работы. Конфигурационные файлы составлены в формате TOML [17]. В таблицах 7, 8, 9 и 10 приводятся описания конфигурационных параметром микросервисов «API шлюз»,

«MonopolyUsersService», «MonopolyLobbiesService» и «MonopolyGameService», соответственно.

Таблица 7 — Конфигурационные параметры «API шлюз»

Параметр	Тип	Описание
HTTP.port	int	Порт работы HTTP сервера
HTTP.isSecure	bool	Флаг режима работы HTTPS
HTTP.Secure.certificate	str	Путь до SSL сертификата
HTTP.Secure.publicKey	str	Путь до публичного TLS ключа
GRPC.ConnectToUsersMicroservice.address	str	IP адрес сервера с «MonopolyUsersService»
GRPC.ConnectToUsersMicroservice.port	int	Порт работы «MonopolyUsersService»
GRPC.ConnectToLobbiesMicroservice.address	str	IP адрес сервера с «MonopolyLobbiesService»
GRPC.ConnectToLobbiesMicroservice.port	int	Порт работы «MonopolyLobbiesService»
Tokens.algorithm	str	Алгоритм JWT
Tokens.secret	str	Секретная фраза JWT

Таблица 8 — Конфигурационные параметры «MonopolyUsersService»

Параметр	Тип	Описание
GRPC.port	int	Порт работы gRPC сервера
Logging.level	str	Уровень ведения журналов
Tokens.algorithm	str	Путь до SSL сертификата
Tokens.secret	str	Путь до публичного TLS ключа
Tokens.accessTokenDuringLife	int	TTL токена доступа (в минутах)
Tokens.refreshTokenDuringLife	int	TTL токена перевыпуска (в днях)
AuthFromVK.getAccessTokenBaseURL	str	URL получения токена VK
AuthFromVK.clientID	str	Идентификатор приложения VK
AuthFromVK.clientSecret	str	Секретная фраза VK
AuthFromVK.redirectURI	str	URL обратного вызова
AuthFromGoogle.getAccessTokenBaseURL	str	URL получения токена Google
AuthFromGoogle.clientSecret	str	Секретная фраза Google
DataBase.name	str	Имя базы данных
DataBase.user	str	Имя пользователя базы данных
DataBase.password	str	Пароль пользователя базы данных
DataBase.host	str	Адрес базы данных
DataBase.port	int	Порт базы данных

Таблица 9 — Конфигурационные параметры «MonopolyLobbiesService»

Параметр	Тип	Описание
GRPC.port	int	Порт работы gRPC сервера
GRPC.ConnectToGameMicroservice.address	str	IP адрес сервера с «MonopolyGameService»
GRPC.ConnectToGameMicroservice.port	int	Порт работы «MonopolyGameService»
Logging.level	str	Уровень ведения журналов
DataBase.name	str	Имя базы данных
DataBase.user	str	Имя пользователя базы данных
DataBase.password	str	Пароль пользователя базы данных
DataBase.host	str	Адрес базы данных
DataBase.port	int	Порт базы данных
LobbyRestrictions.minMaxPlayers	str	Минимально возможное значение кол-ва игроков
LobbyRestrictions.maxMaxPlayers	str	Максимально возможное значение кол-ва игроков
LobbyRestrictions.minTimeForTurn	str	Минимально возможное значение времени на ход
LobbyRestrictions.maxTimeForTurn	str	Максимально возможное значение времени на ход
LobbyRestrictions.minScoreVictoryValue	str	Минимально возможное значение счёта для победы
LobbyRestrictions.maxScoreVictoryValue	str	Максимально возможное значение счёта для победы
LobbyRestrictions.minTurnVictoryValue	str	Минимально возможное значение ходов до победы
LobbyRestrictions.maxTurnVictoryValue	str	Максимально возможное значение ходов до победы

Таблица 10 — Конфигурационные параметры «MonopolyGamesService»

Параметр	Тип	Описание
GRPC.port	int	Порт работы gRPC сервера
GRPC.ConnectToLobbiesMicroservice.address	str	IP адрес сервера с «MonopolyLobbiesService»
GRPC.ConnectToLobbiesMicroservice.port	int	Порт работы «MonopolyLobbiesService»
Logging.level	str	Уровень ведения журналов
GameParams.sabotageImpactMultiplier	float	Множитель ущерба от диверсии
GameParams.Economy.startingBalance	int	Начальное значение баланса
GameParams.Economy.BaseCosts.sawmill	int	Базовая стоимость лесопилки
GameParams.Economy.BaseCosts.coalStation	int	Базовая стоимость ТЭС
GameParams.Economy.BaseCosts.atomicStation	int	Базовая стоимость АЭС
GameParams.Economy.BaseCosts.forest	int	Базовая стоимость леса
GameParams.Economy.BaseCosts.coal	int	Базовая стоимость угольного месторождения
GameParams.Economy.BaseCosts.uranium	int	Базовая стоимость уранового месторождения

Окончание таблицы 10

Параметр	Тип	Описание
GameParams.Economy.BaseCosts.sabotage	int	Базовая стоимость диверсии
GameParams.Economy.BaseCosts.arrow	int	Базовая стоимость выбора пути
GameParams.Economy.BaseIncomes.sawmill	str	Базовая доход лесопилки
GameParams.Economy.BaseIncomes.coalStation	int	Базовая доход ТЭС
GameParams.Economy.BaseIncomes.atomicStation	str	Базовая доход АЭС

3.4 Выводы по главе

В данной главе были описаны инструменты, использованные в процессе разработки системы и структура каждого микросервиса. Помимо этого, описан процесс тестирования, состоящий из ручных и нагрузочных тестов. Приводятся возможные варианты развёртывания системы и описания конфигурационных параметров разработанных микросервисов.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы было выполнено проектирование серверной системы для многопользовательской игры «Монополия» с использованием принципов микросервисной архитектуры и современных подходов. В процессе проектирования была разработана архитектурная схема, описаны HTTP методы, описаны способы и методы межкомпонентного взаимодействия, разработан ряд UML диаграмм.

На основании результатов проектирования была разработана серверная система, состоящая из нескольких компонент — микросервисов. Для каждого микросервиса выбраны подходящие современные средства разработки.

На фоне аналогов разработанная серверная систем, вкупе с клиентским приложением имеет ряд характерных особенностей, уникальную реализацию игрового процесса, систему игровых комнат и рейтинговых игр.

Выбранный тип архитектуры разработанного приложения позволит легко проводить дальнейшую разработку.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 GitHub: The complete developer platform to build, scale, and deliver secure software. [Электронный ресурс]. – 2023. – URL: https://github.com.com (дата обращения 01.03.2023).

2 McIlroy M.D «UNIX Time-Sharing System»// The Bell System Technical Journal vol. 57, no. 6, part 2: – 1978. – С. 1905–1929.

3 Protocol Buffers : Google Developers : сайт — URL https://developers.google.com/protocol-buffers (дата обращения 05.01.2023).

4 Фреймворк gRPC / Документация gRPC : сайт. – URL: https://grpc.io/docs/ (дата обращения: 23.03.2023).

5 OAuth 2.0 / The industry-standard protocol for authorization : сайт. – URL: https://oauth.net/2/ (дата обращения: 17.05.2023).

6 VK для разработчиков / Authorization Code Flow для получения ключа доступа пользователя : сайт. – URL: https://dev.vk.com/api/access-token/authcode-flow-user (дата обращения: 17.05.2023).

7 Google Identity / Using OAuth 2.0 to Access Google APIs : сайт. – URL: https://developers.google.com/identity/protocols/oauth2 (дата обращения: 17.05.2023).

8 RFC 7519/ JSON Web Token : сайт. – URL: https://www.rfc-editor.org/rfc/rfc7519 (дата обращения: 18.05.2023).

9 RFC 4868/ undefined Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 with IPsec : сайт. – URL: https://www.rfc-editor.org/rfc/rfc4868 (дата обращения: 18.05.2023).

10 Git / A free and open source distributed version control system : сайт. – URL: https://git-scm.com/ (дата обращения: 18.05.2023).

11 Golang / undefined Build simple, secure, scalable systems with Go : сайт. – URL: https://go.dev/ (дата обращения: 18.05.2023).

12 Репозиторий «SergeyZvyagin/monopoly-api-gateway» / GitHub : сайт. – URL: https://github.com/SergeyZvyagin/monopoly-gateway (дата обращения: 17.05.2023).

13 Репозиторий «SergeyZvyagin/monopoly-game-service» / GitHub : сайт. – URL: https://github.com/SergeyZvyagin/monopoly-game-service (дата обращения: 17.05.2023).

14 Репозиторий «SergeyZvyagin/monopoly-users-service» / GitHub : сайт. – URL: https://github.com/SergeyZvyagin/monopoly-users-service (дата обращения: 18.05.2023).

15 Репозиторий «SergeyZvyagin/monopoly-lobbies-service» / GitHub : сайт. – URL: https://github.com/SergeyZvyagin/monopoly-lobbies-service (дата обращения: 19.05.2023).

16 Docker / Accelerated, Containerized Application Development : сайт. – URL: https://www.docker.com/ (дата обращения: 19.05.2023).

17 TOML / A config file format for humans : сайт. – URL: https://toml.io/en/ (дата обращения: 30.05.2023).

ПРИЛОЖЕНИЕ А

Правила и особенности игры

Начало игры

Игра начинается с определения сервером порядка ходов участников случайным образом — это соответствует назначению игрокам цветов фишек, которые изображены на рисунке А.1 — они отображают расположение игроков на карте. В начале игры фишки всех игроков располагаются на первом поле. Сервер размещает особые поля карты посредством случайной генерации.

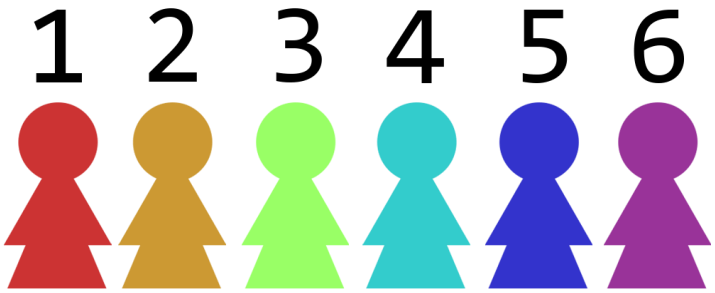

Рисунок А.1 — Цвета фишек игроков

Ход игрока

В начале хода игрока бросаются две игральные кости, затем их значения складываются, и фишка играющего перемещается на соответствующее поле, номер которого определяется суммой номера предыдущей клетки и суммой выпавших значений кубиков (данный процесс показан на рисунке А.2). Если поле, на котором находится фишка игрока, является активируемым, участнику даётся фиксированное время на ход — за это время игрок имеет право применить действие активной клетки, либо досрочно завершить свой ход.

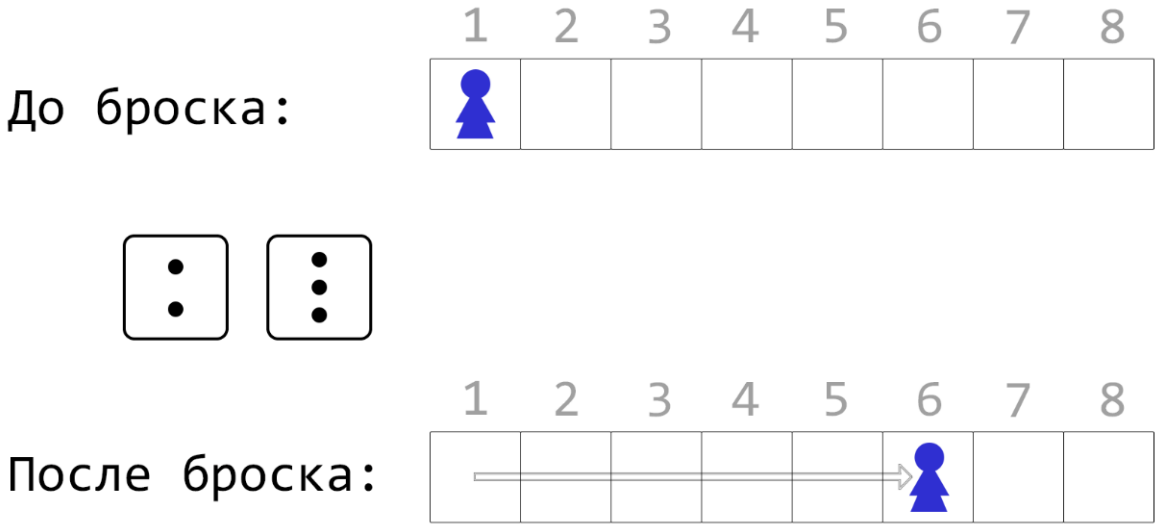

Рисунок А.2 — Перемещение фишки игрока

Типы полей

В игре предусмотрены поля разного типа (все поля представлены на рисунке А.10):

– начальное поле (изображено на рисунке А.3): поле, с которого начинают игру все участники. При попадании фишки игрока на данное поле повторно в течение игры, ему полагается денежный бонус. Количество полей на карте: 1;

Рисунок А.3 — Вид начального поля

– обычное поле (изображено на рисунке А.4): не имеет влияния на игровой процесс;

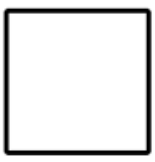

Рисунок А.4 — Вид обычного поля

– поле с активом (изображено на рисунке А.5): имеет активное действие, позволяющее игроку приобрести бизнес за определённую цену. Количество полей на карте: 8 с лесопилками, 4 с ТЭС, 3 с АЭС;

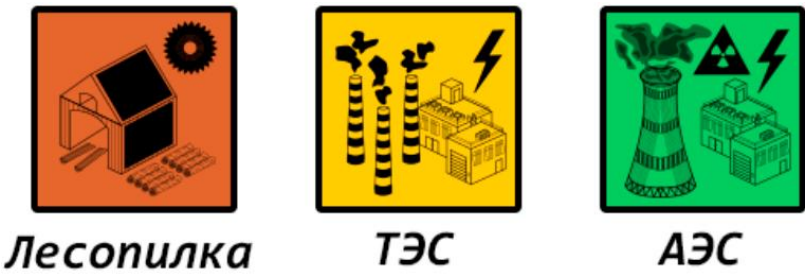

Рисунок А.5 — Вид полей с активами трёх типов

– поле с ресурсом (изображено на рисунке А.6): имеет активное действие, позволяющее игроку развернуть на данном поле свою добывающую станцию. Количество полей на карте: по 3 поля на каждый тип ресурса;

Рисунок А.6 — Вид полей с ресурсами трёх типов

– поле с отпуском (изображено на рисунке А.7): игрок, фишка которого попала на данное поле, вынужден пропустить свой ход в следующей очереди ходов, при этом ему полагается денежный бонус. Количество полей на карте: 2;

Рисунок А.7 — Вид поля с отпуском

– поле с диверсией (изображено на рисунке А.8): имеет активное действие, позволяющее игроку выбрать клетку с занятой другим участником добывающей станцией или бизнесом, которая будет освобождена от владения, при этом данную клетку можно будет приобрести забесплатно любому участнику. Кроме этого, цель диверсии (игрок, потерявший актив или станцию) лишается 5% своего текущего денежного запаса. После использования поле с диверсией перемещается на место случайного обычного поля. Количество полей на карте: 3.

Рисунок А.8 — Вид поля с диверсией

– поле со стрелкой (изображено на рисунке А.9): определяет, в какую сторону пойдёт фишка игрока, если путь лежит через неё. После прохождения через данное поле игрок за определённую плату может выбрать, в какую именно сторону будет повёрнута стрелка. В случае, если выбор не будет сделан, указатель повернётся по часовой стрелке.

Рисунок А.9 — Вид поля со стрелкой

Рисунок А.10 — Все виды полей

Активы

В игре можно купить активы трёх видов: лесопилка, тепловая электрическая станция (ТЭС) и атомная электрическая станция (АЭС); среди

перечисленных активов, лесопилки приносят наименьший доход, а АЭС — наибольший.

Каждый ход актив под владением игрока приносит ему прибыль. При попадании фишки игрока на уже купленный им актив, он получает дополнительную выплату в размере 0,5 от прибыли, которую производит данный актив. Если фишка игрока переместилась на актив, занятый другим игроком, он выплачивает последнему налог в размере 3% от своего текущего денежного запаса.

Ресурсы

Существует зависимость производств от следующих ресурсов: лесных массивов, залежей каменного угля и залежей урана соответственно списку активов. Зависимость выражается в следующем:

– пример №1: игрок А имеет в своём распоряжении три атомных электрических станции. Установив добывающую станцию урана, он повышает прибыль каждого из своих трёх активов на 25%, в сумме увеличивая свой доход от АЭС на 75%. Приведённый пример представлен на рисунке А.11;

Рисунок А.11 — Зависимость активов от ресурсов (пример №1)

– пример №2: игроки А и Б в сумме имеют в своём распоряжении три тепловые электрические станции. Игрок В, установив добывающую станцию каменного угля, будет получать часть прибыли (по 25% за ТЭС) активов игроков А и Б. Это, в свою очередь, значит, что игроки А и Б будут получать от своих ТЭС на 25% меньше прибыли. Пример проиллюстрирован на рисунке А.12.

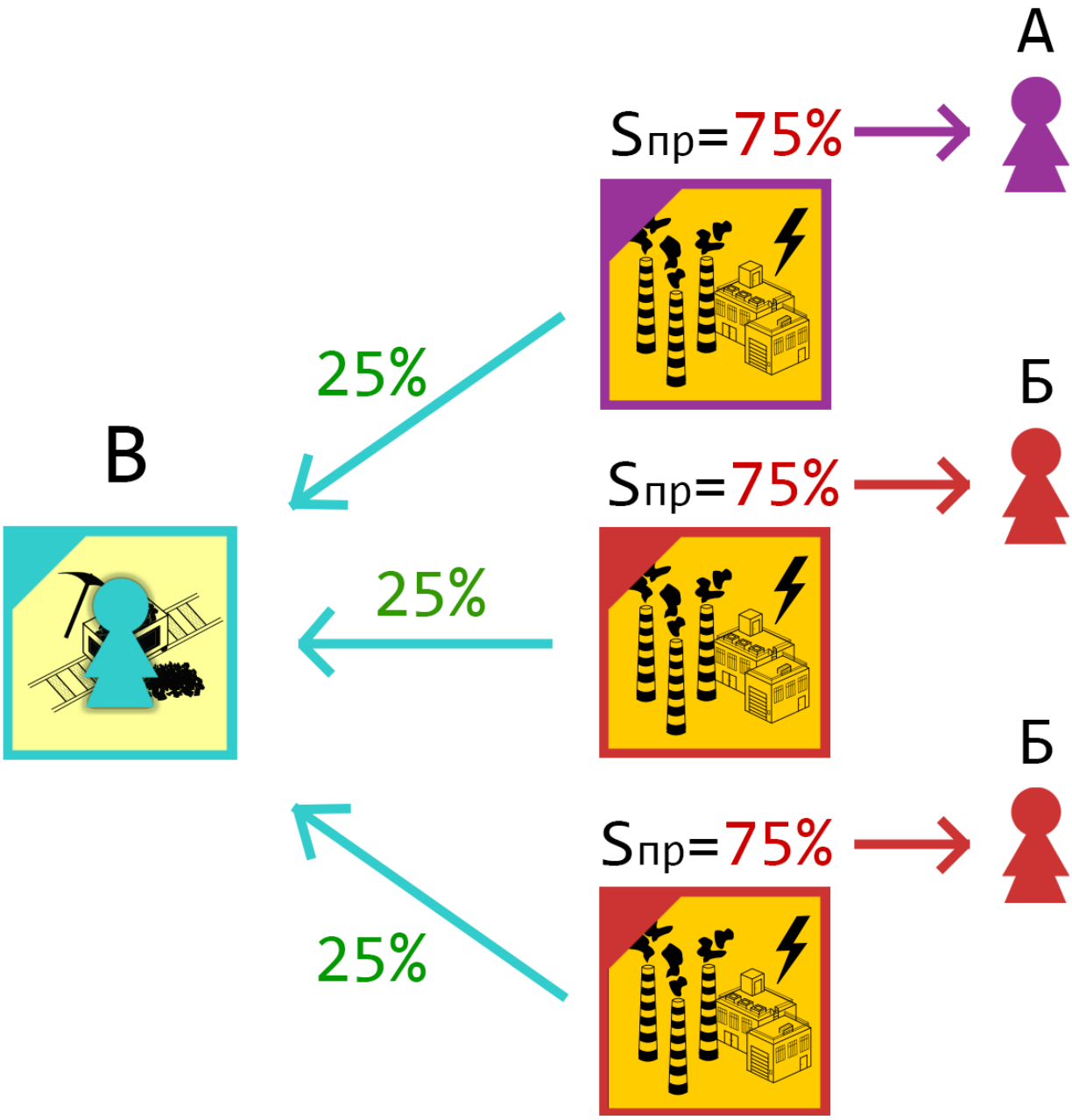

Рисунок А.12 — Зависимость активов от ресурсов (пример №2)

Игрок, фишка которого попала на поле с активом другого участника, может перекупить это поле в своё владение, если у него есть сумма, равная значению, вычисляемому по формуле (А.1).

$$S = 1{,}5 \cdot S_0 + \mathrm{k} \cdot \left\lceil \left| \frac{S_{\text{Д.и.п.}} - S_{\text{Д.и.в.}}}{k} \right|^{1{,}2} \right\rceil, \qquad \text{(А.1)}$$

где S — сумма, которую платит перекупающий игрок;

S_0 — базовая стоимость актива;

$S_{Д.и.п.}$ — доход перекупающего игрока;

$S_{Д.и.в.}$ — доход игрока-владельца актива;

k — регулирующий коэффициент, рассчитывающийся по формуле (А.2).

$$k = 10^{digits\left(avg\left(S_{Д.и.п.}, S_{Д.и.в.}\right)\right) - 1}, \quad (А.2)$$

где k — регулирующий коэффициент;

digits(x) — функция, возвращающая число разрядов в целой части числа x (например: digits(1000,294) = 4; digits(29492) = 5);

avg(x, y) — функция, возвращающая среднее значение между числами x и y;

$S_{Д.и.п.}$ — доход перекупающего игрока;

$S_{Д.и.в.}$ — доход игрока-владельца актива;

В таком случае 50% этих денег перейдёт бывшему владельцу производства.

Стадии игры

В игре предусмотрено три стадии игры:

– первая стадия. Ходы 1-24. Данная стадия характеризуется тем, что на карте имеются поля со всеми активами, кроме активов атомных электрических станций. Кроме полей с активами присутствуют поля с отпуском;

– вторая стадия. Ходы 25-48. Карта дополняется вертикальным мостом посередине. На клетках-развилках размещаются особые поля со стрелками. На месте некоторых обычных полей появляются поля со всеми ресурсами, кроме залежей урана;

– третья стадия. Ходы 49-72 (либо меньше или больше 72 — зависит от настройки в пользовательской игровой комнате). Карта дополняется

горизонтальным мостом посередине, в итоге в центре карты формируется перекрёсток. На месте некоторых обычных полей появляются поля с диверсиями, ресурсами урана, и активами атомных электрических станций.

Процесс изменения карты продемонстрирован на рисунке А.13.

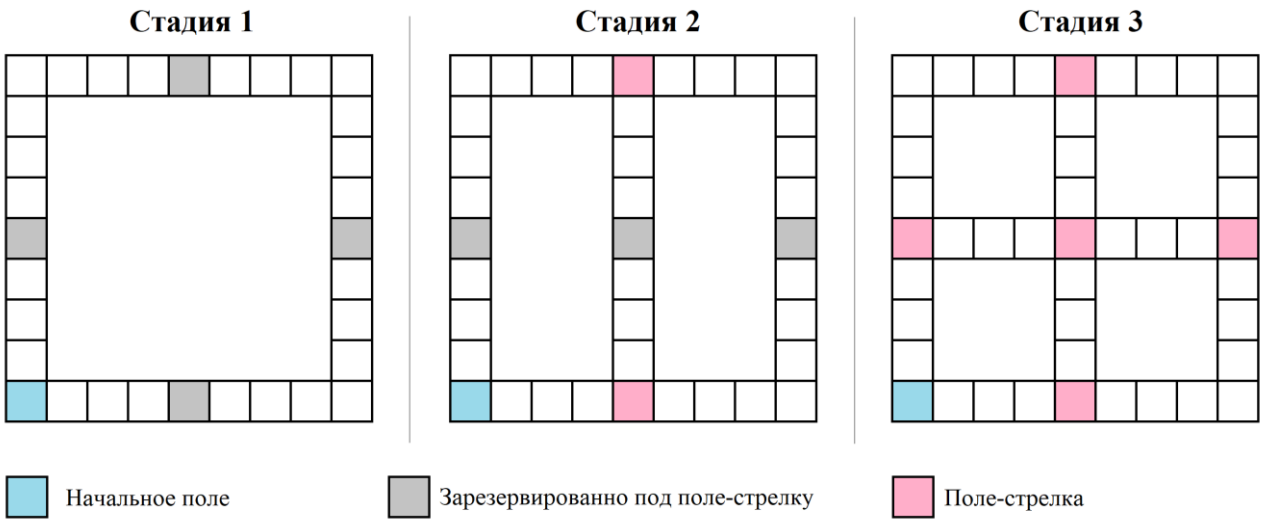

Рисунок А.13 — Расширение карты по стадиям

Конец игры

Предусмотрено несколько сценариев окончания игры и определения победителя:

– по балансу: победителем считается игрок, первым набравший установленную в настройках сумму для победы на своём счёте;

– по балансу игроков на последний ход (если игра ограничена числом ходов): победителем считается игрок, имеющий наибольшую сумму на счету;

– по последнему оставшемуся игроку: если все игроки, кроме одного, сдались или отключились без дальнейшего присоединения к игре, то оставшийся игрок назначается победителем, несмотря на свой денежный баланс.

Возможность сдаться и отключение игрока

В обычном матче предусмотрена возможность сдаться. Если игрок воспользовался данной возможностью, то его денежный баланс будет разделён поровну между оставшимися участниками.

При незапланированном отключении игрока от матча устанавливается пауза на одну минуту, предоставляющая игроку небольшое количество времени для возвращения в игру. По истечению данного времени игра продолжается, а фишкой отключившегося игрока управляет «бот», однако игрок всё ещё сможет вернуться в игру, пока не прошло 10 общих ходов с момента его отключения. На 11 ход игрок будет отключен от матча, его фишка будет удалена из игры, купленные им клетки — освобождены, и его денежный баланс так же будет разделён поровну между оставшимися участниками, как это работает при использовании игроком возможности сдаться.

Для каждого игрока пауза устанавливается единожды, то есть при повторном отключении игрока после его возвращения игра уже не будет приостановлена. Ходы «бота», оставшиеся до отключения, не восполняются при повторном подключении игрока к матчу.

«Бот»

«Бот» служит для управления в течение 10 общих ходов фишками игроков, которые отключились от игры ввиду незапланированной ситуации.

Если его фишка попала на активируемое поле, то «бот» по умолчанию скупает свободные активы, но игнорирует возможные действия любых других полей. «Бот» не способен перекупать чужие активы. Он не является кандидатом на победу, даже если его сумма на счету достигнет значения, необходимого для победы, или если в игре наступил последний ход, баланс «бота» в таблице игроков игнорируется. Однако, стоит учитывать, что если денежный баланс

«бота» достигнет значения, достаточного для завершения игры, это инициализирует конец игры, и его баланс будет игнорироваться.

Проводить диверсию по отношению к «боту» можно, так как он просто является заместителем игрока, который отключился. По аналогии, его активы также можно перекупать.

Рейтинговый матч

Рейтинговая игра начинается только с четырьмя игроками. В рейтинговом матче отсутствует возможность сдаться. Отключившийся игрок, не присоединившийся далее по игре в течение 10 общих ходов с момента отключения, на 11-тый ход уже не сможет подключиться обратно и получит штраф в размере «-10» RP («Rating Points» — «Баллы Рейтинга»).

Настройки рейтинговой игры фиксированные:

– конец игры по ходам: 72 хода;

– конец игры по балансу: не ограничено;

– время на ход: 8 секунд;

Количество очков RP, на которое изменяется рейтинг игроков по окончанию игры, определяется занятым местом:

– первое место: +10 RP;

– второе место: +5 RP;

– третье место: -5 RP;

– четвёртое место: -10 RP.

ПРИЛОЖЕНИЕ Б

Образцы JSON структур тел HTTP запросов и ответов

Структура «tokens»

```
{
   "tokens": {
      "accessToken": "abcd.abcd.abcd",
      "refreshToken": "dcba.dcba.dcba"
   }
}
```

Структура «userInfo»

```
{
   "userInfo": {
      "ID": 1,
      "nickname": "Bob",
      "isGuest": false,
      "rating": 0
   }
}
```

Структура «settings»

```
{
   "settings": {
      "name": "The Bob’s lobby",
      "type": "PRIVATE",
```

```
        "password": "1234",
        "maxPlayers": 6
        "timeForTurn": 8,
        "victoryType": "BOTH",
        "scoreVictoryValue": 2000,
        "turnVictoryValue": 72
    }
}
```


Структура «lobbyInfo»


```
{
    "lobbyInfo": {
        "lobbyID": 1,
        "ownerID": 1,
        "settings": {
            "name": "The Bob’s lobby",
            "type": "PRIVATE",
            "password": "1234",
            "maxPlayers": 6,
            "timeForTurn": 8,
            "victoryType": "BOTH",
            "scoreVictoryValue": 2000,
            "turnVictoryValue": 72
        },
        "players": [
            {
                "userInfo": {
                    "ID": 1,
```

```
                "nickname": "Bob",
                "isGuest": false,
                "rating": 0
            },
            "isReady": false
        }
    ],
    "isRunning": true,
    "connection": {
        "address": "172.23.0.100",
        "port": 53111
    }
  }
}
```


Структура «lobbies»


```
{
  "lobbies": [
    {
      "lobbyID": 1,
      "name": "The Bob’s lobby",
      "isPrivate": true,
      "maxPlayers": 6,
      "playersNow": 1
    }
  ]
}
```

Структура «connection»

```
{
   "connection": {
      "address": "172.23.0.100",
      "port": 53111
   }
}
```

ПРИЛОЖЕНИЕ В

Proto структура «UsersMicroservice»

```
syntax = "proto3";

package UsersMicroservice;

option go_package = "./users_pb";

service UsersService {
   rpc AuthFromVK(AuthRequest) returns (AuthResponse);
   rpc AuthFromGoogle(AuthRequest) returns (AuthResponse);
   rpc AuthAsGuest(AuthRequest) returns (AuthResponse);
   rpc RefreshAccessToken(RefreshAccessTokenRequest) returns
(RefreshAccessTokenResponse);
   rpc GetInfo(GetInfoRequest) returns (GetInfoResponse);
   rpc ChangeNickname(ChangeNicknameRequest) returns
(ChangeNicknameResponse);
}

enum ExitStatus {
   SUCCESS = 0;
   CODING_ERROR = 1;
   DECODING_ERROR = 2;
   FAILED_DEPENDENCY = 3;
   RESOURCE_NOT_AVAILABLE = 4;
}

message TokenPair {
```

```
    string accessToken = 1;
    string refreshToken = 2;
}

message User {
    uint32 ID = 1;
    string nickname = 2;
    bool isGuest = 3;
    uint32 rating = 4;
}


message AuthRequest {
    string authCode = 1;
}

message AuthResponse {
    ExitStatus status = 1;
    optional TokenPair tokens = 2;
    optional User userInfo = 3;
}

message ChangeNicknameRequest {
    uint32 requesterID = 1;
    string newNickname = 3;
}

message ChangeNicknameResponse {
    ExitStatus status = 1;
```

```
}

message RefreshAccessTokenRequest {
   uint32 requesterID = 1;
}

message RefreshAccessTokenResponse {
   ExitStatus status = 1;
   optional string accessToken = 2;
}

message GetInfoRequest {
   uint32 requesterID = 1;
   uint32 userID = 2;
}

message GetInfoResponse {
   ExitStatus status = 1;
   optional User userInfo = 2;
}
```

ПРИЛОЖЕНИЕ Г

Proto структура «LobbiesMicroservice»

```
syntax = "proto3";

package LobbiesMicroservice;

import "UsersMicroservice.proto";

option go_package = "./lobbies_pb";

service LobbiesService {
    rpc Create(SetSettingsRequest) returns (AboutLobbyResponse);
    rpc GetList(RequesterOnlyRequest) returns (GetListResponse);
    rpc Connect(ConnectRequest) returns (AboutLobbyResponse);
    rpc GetInfo(RequesterOnlyRequest) returns (AboutLobbyResponse);
    rpc UpdateSettings(SetSettingsRequest) returns (StatusOnlyResponse);
    rpc ConnectToRanked(RequesterOnlyRequest) returns
(AboutLobbyResponse);
    rpc Disconnect(RequesterOnlyRequest) returns (StatusOnlyResponse);
    rpc SwitchReadiness(RequesterOnlyRequest) returns (StatusOnlyResponse);
    rpc RaisePlayer(RequesterAndTargetRequest) returns
(StatusOnlyResponse);
    rpc KickPlayer(RequesterAndTargetRequest) returns (StatusOnlyResponse);
    rpc CheckInActive(RequesterOnlyRequest) returns
(CheckInActiveResponse);
    rpc Run(RequesterOnlyRequest) returns (StatusOnlyResponse);
    rpc Delete(RequesterOnlyRequest) returns (StatusOnlyResponse);
```

```
        rpc RegisterPlayerConnection(RequesterOnlyRequest) returns
(StatusOnlyResponse);
        rpc RegisterPlayerDisconnection(RequesterOnlyRequest) returns
(StatusOnlyResponse);
        rpc RegisterGameResults(RegisterResultsRequest) returns
(StatusOnlyResponse);
    }

    enum ExitStatus {
        SUCCESS = 0;
        CODING_ERROR = 1;
        DECODING_ERROR = 2;
        FAILED_DEPENDENCY = 3;
        RESOURCE_NOT_AVAILABLE = 4;
        DB_LEVEL_ERROR = 5;
        BAD_VALUE = 6;
        FORBIDDEN = 7;
    }

    enum VictoryType {
        SCORE = 0;
        TURN = 1;
        BOTH = 2;
    }

    enum LobbyType {
        PUBLIC = 0;
        PRIVATE = 1;
        RANKED = 2;
```

```
}

message LobbySettings {
    string name = 1;
    LobbyType type = 2;
    optional string password = 3;
    uint32 maxPlayers = 4;
    uint32 timeForTurn = 5;
    VictoryType victoryType = 6;
    uint32 scoreVictoryValue = 7;
    uint32 turnVictoryValue = 8;
}

message PlayerResult {
    uint32 userID = 1;
    uint32 placement = 2;
}

message Player {
    UsersMicroservice.User userEntity = 1;
    bool isReady = 2;
}

message GameConnectionInfo {
    string address = 1;
    uint32 port = 2;
}

message ShortCurrentLobbyInfo {
```

```
    uint32 lobbyID = 1;
    string name = 2;
    bool isPrivate = 3;
    uint32 maxPlayers = 4;
    uint32 playersNow = 5;
}

message FullCurrentLobbyInfo {
    uint32 lobbyID = 1;
    uint32 ownerID = 2;
    LobbySettings settings = 3;
    repeated Player players = 4;
    bool timerIsActivate = 5;
    optional GameConnectionInfo connection = 6;
}


// RPCs
// Shared messages
message AboutLobbyResponse {
    ExitStatus status = 1;
    optional FullCurrentLobbyInfo lobbyInfo = 2;
}

message RequesterOnlyRequest {
    uint32 requesterID = 1;
}

message RequesterAndTargetRequest {
```

```
    uint32 requesterID = 1;
    uint32 targetPlayerID = 2;
}

message StatusOnlyResponse {
    ExitStatus status = 1;
}


// RPC Create
message SetSettingsRequest {
    uint32 requesterID = 1;
    optional LobbySettings settings = 2;
} // Response is AboutLobbyResponse

// RPC Connect
message ConnectRequest {
    uint32 requesterID = 1;
    uint32 lobbyID = 2;
    optional string password =3;
} // Response is AboutLobbyResponse

// RPC GetList
message GetListResponse {
   ExitStatus status = 1;
   repeated ShortCurrentLobbyInfo lobbies = 2;
} // Request is RequesterOnlyRequest

message CheckInActiveResponse {
```

```
    ExitStatus status = 1;
    optional GameConnectionInfo connection = 2;
}

message RegisterResultsRequest {
    repeated PlayerResult result = 1;
```

ПРИЛОЖЕНИЕ Д

Proto структуры «GameMicroservice» и «MonopolyGameMessages»

Proto структура «GameMicroservice»

```
syntax = "proto3";

package GameMicroservice;

import "LobbiesMicroservice.proto";

option go_package = "./game_pb";

service GameMicroservice {
   rpc Run(LobbiesMicroservice.FullCurrentLobbyInfo) returns
(LobbiesMicroservice.GameConnectionInfo);
}
```

Proto структура «MonopolyGameMessages»

```
syntax = "proto3";

package MonopolyGameMessages;

option go_package = "./game_cs_pb";


// Enumerations
enum AwaitedMessage {
```

```
    STAGE_IS_DISPLAYED = 0;
    DICES_ARE_DRAWN = 1;
    PIECE_MOVING_IS_DRAWN = 2;
    TIME_FOR_RESOLVE_OUT = 3;
    BUY_RESOLVE = 4;
    DO_SABOTAGE_RESOLVE = 5;
    BUY_ARROW_RESOLVE = 6;
    TIME_FOR_WAIT_OUT = 7;
}

enum TileType {
    VOID = 0;
    COMMON = 1;
    BEGINNING = 2;
    SAWMILL = 3;
    COAL_STATION = 4;
    ATOMIC_STATION = 5;
    FOREST = 6;
    COAL = 7;
    URANIUM = 8;
    VACATION = 9;
    SABOTAGE = 10;
    ARROW = 11;
}

enum ArrowDirection {
    UP = 0;
    RIGHT = 1;
    DOWN = 2;
```

```
    LEFT = 3;
}

enum PlayerNumber {
    NONE = 0;
    PLAYER1        = 1;
    PLAYER2 = 2;
    PLAYER3 = 3;
    PLAYER4 = 4;
    PLAYER5 = 5;
    PLAYER6 = 6;
}

enum State {
    WAITING = 0;
    RELAX = 1;
    PLAYER_TURN = 2;
    END = 1;
}

// Main messages
message ClientHello {
    uint32 id = 1;
}

message ServerHello {
    uint32 lobbyID = 1;
    string lobbyName = 2;
    uint32 timeForResolve = 3;
```

```
    repeated Player players = 4;
    repeated TileStat baseCost = 5;
    repeated TileStat baseIncome = 6;
    GameState currentGameState = 7;
}

message StateRefresh {
    GameState currentGameState = 1;
    optional AwaitedResolve awaitedResolve = 2;
    optional AwaitedMessage awaitedSignal = 3;
}


// Resolve messages
message BuyResolve {
    bool isBought = 1;
}

message DoSabotageResolve {
    bool isSabotaged = 1;
    optional uint32 tileOrder = 2;
}

message BuyArrowResolve {
    bool isBought = 1;
    optional ArrowDirection newDirection = 2;
}
```

```
// Awaited signal messages
message StageIsDisplayed {}
message DicesAreDrawn {}
message PieceMovingIsDrawn {}
message TimeForResolveOut {}
message TimeForWaitOut {}


// Additional messages
message GameState {
   State state = 1;
   uint32 currentTurn = 2;
   uint32 currentStage = 3;
   optional PlayerNumber whoseTurn = 4;
   repeated uint32 trace = 5;
   Dices dices = 6;
   repeated Tile map = 7;
   repeated Player players = 8;
}

message AwaitedResolve {
   AwaitedMessage await = 1;
   Tile target = 2;
}

message TileStat {
   TileType type = 1;
   uint32 value = 2;
}
```

```
message Dices {
    uint32 left = 1;
    uint32 right = 2;
}

message Tile {
    uint32 order = 1;
    TileType type = 2;
    PlayerNumber owner = 3;
    uint32 minDisplayStage = 4;
    uint32 minActivationStage = 5;
    repeated PlayerNumber piecesOnPosition = 6;
    uint32 costForYou = 7;
    uint32 incomeForOwner = 8;
    optional ArrowDirection direction = 9;
    repeated ArrowDirection allowDirections = 10;
}

message Player {
    uint32 userID = 1;
    string nickname = 2;
    PlayerNumber order = 3;
    uint32 income = 4;
    uint32 balance = 5;
}
```

Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение высшего образования
«**СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**»

Институт космических и информационных технологий

Кафедра вычислительной техники

УТВЕРДЖАЮ
Заведующий кафедрой
_____ О. В. Непомнящий
подпись
«22.» 06 2023 г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 — Информатика и вычислительная техника

Игра «Монополия». Серверная часть

Руководитель _____ 16.6.23 ст. преподаватель В. С. Васильев
подпись, дата

Выпускник _____ 16.06.2023 С. А. Звягин
подпись, дата

Нормоконтролер _____ 16.6.23 ст. преподаватель В. С. Васильев
подпись, дата

Красноярск 2023