

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

Кафедра

вычислительной техники

«Вычислительная техника»

Специальность: 09.05.01 «Применение и эксплуатация
автоматизированных систем специального
назначения»
Специализация: «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

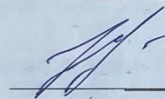
ДИПЛОМНЫЙ ПРОЕКТ

на тему

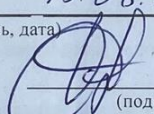
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ПЕРЕДАЧИ

ПОТОКОВЫХ ДАННЫХ

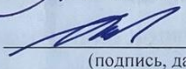
Студент

 19.06.23 Новинская Алина Михайловна
(подпись, дата) (ФИО полностью)

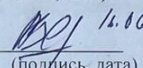
Руководитель

 19.06.23 Федюнин Р.Н.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр

 Бычков А.С. 19.06.2023
(подпись, дата) (фамилия, инициалы)

Рецензент

Кафедра «Вычислительная техника» Фак-И
ПАО «Роснефть» Волга-1, Волга-2
(должность, место работы)  14.06.2023 Себастьян П.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 2.06.2023 г. № 14)

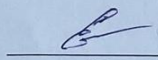
Заведующий кафедрой

 Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой отлично

(протокол заседания ГЭК от 19.06.2023 № 6)

Секретарь ГЭК

 Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2023

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

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Митрохин М.А.
«26» апреля 2023 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Новинская Алина Михайловна гр. 18ВОЭ1 факультета вычислительной техники.

2. Тема работы: «Программное обеспечение передачи потоковых данных».

Тема утверждена приказом ПГУ № 458/о от 26.04.2023 г.

3. Руководитель работы Федюнин Р.Н.

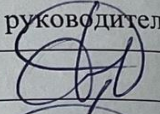
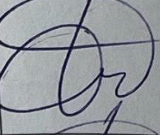
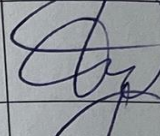
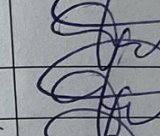
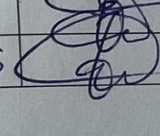
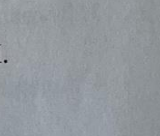
4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – Программа предназначена для видео и аудио контента в режиме клиент сервер. В программе необходимо предусмотреть функции хранения данных в СУБД. Программа должна работать в рамках сети учреждения в двух режимах: проверка канала связи и режим конференции.

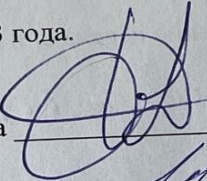
5. Перечень подлежащих разработке вопросов.

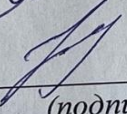
Требуется произвести анализ предметной области, определить функциональный состав программы, выбрать платформу и стек для реализации клиентского приложения, разработать алгоритм работы программы, разработать UML диаграммы, с последующим написанием программы.

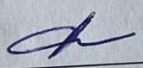
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	26.04.23 - 28.04.23.	
2.	Выбор свободно распространяемой платформы для работы с потоковыми данными	10	29.04.23 - 30.04.23	
3.	Выбор и апробация стека технологий для реализации клиентской программы	20	31.04.23 - 10.05.23	
4.	Проектирование и реализация базы данных	10	11.05.23 - 13.05.23.	
5.	Реализация клиентской программы	10	14.05.23 - 19.05.23	
6.	Отладка системы	10	20.05.23 - 26.05.23	
7.	Оформление выпускной работы	30	27.05.23 - 2.06.23	

Дата выдачи задания « » 2023 года.

Руководитель дипломного проекта  Федюнин Р. Н.
(подпись, дата)

Задание к исполнению принял студент  19.06.23 Новинская А.М.
(подпись, дата)

Работу к защите допустить
Декан факультета  19.06.2023 Фионова Л.Р.
(подпись, дата)

Реферат

Работа содержит 94 лист., 2 рис., 13 табл., 6 ист., 7 прилож.
ПЕРЕДАЧА ИНФОРМАЦИИ ПО СЕТИ, ОБРАБОТКА ИНФОРМАЦИИ,
ВОСПРОИЗВЕДЕНИЕ ВИДЕО, КЛИЕНТСЕРВЕРНАЯ АРХИТЕКТУРА,
WebRTC

Цель ВКР – разработать приложение «Программное обеспечение передачи потоковых данных», позволяющее пользователям взаимодействовать между собой посредством обмена текстовой, видео и голосовой информации.

В ходе выполнения ВКР были разработаны:

Клиентское приложение для формирования информации в виде текстовых, голосовых и/или видео потока и передачи сформированных потоков данных через сервер другим клиентам.

Серверное приложение для приёма, обработки и отправки звуковых и текстовых данных, полученных от клиентов.

В результате проделанной работы разработано клиент-серверное приложение, полностью соответствующее поставленной задаче, а именно:

- разработано серверное приложение, которое принимает звуковую и текстовую информацию от клиентов;
- обрабатывает её и отправляет клиентским приложениям;
- разработано клиентское приложение с графическим интерфейсом позволяющее отправлять на сервер, принимать с сервера и выводить на устройства вывода звуковую и текстовую информацию.

					ПГУ1.09.05.01.08.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение передачи потоковых данных Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Новинская А.М.							
Провер.		Федюнин Р.Н.							
Н. Контр.		Бычков А.С.					2	94	
						ФВТ, каф. ВТ			
Утверд.		Митрохин М.А.				гр.18ВОЭ1			

Содержание

Оглавление

Перечень сокращений и обозначений.....	5
Введение	6
1. Предпроектный анализ предметной области	8
1.1 Анализ предметной области	8
1.2 Техническое задание	12
1.2.1 Назначение разработки	12
1.2.2.2 Требования к составу и параметрам технических средств	13
1.3 Стадии и этапы разработки	13
1.4 Анализ стека технологий	14
1.4.4 Протокол и платформа WebRTC	17
1.6 Выбор языка программирования.....	22
1.7 Выбор системы управления базой данных	23
1.8 Выводы по первой главе	23
2. Проектирование и реализация программного обеспечения передачи потоковых данных.	24
2.1 Проектирование и реализация серверной части приложения	25
2.1.1 Режим работы «проверка канала связи»	25
2.1.2 Режим работы «конференция»	27
2.2 Проектирование клиентской части (интерфейса)	31
2.3 Проектирование базы данных приложения	36
3. Технология развертывания сервера программного обеспечения передачи потоковых данных.	39
3.1 Локальная установка медиа сервера на Ubuntu	39
3.2 Установка медиа сервера с помощью образа Docker	40
3.3 Выводы по третьей главе	43
4. Экономическое обоснование эффективности разработки и внедрения программы потоковой обработки данных	44
5. Безопасность жизнедеятельности. Особенности микроклимата на рабочем месте оборудованном ЭВМ.	50

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Закключение.....	53
Список использованных источников	54
Приложение А	55
(обязательное)	55
Приложение А.1 – Диаграмма вариантов	56
Приложение А.2 – Диаграмма развертывания	57
Приложение А.3 – Диаграмма деятельности.....	58
Приложение А.4 – Диаграмма деятельности.....	59
Приложение А.5 – Диаграмма данных	60
Приложение Б.....	61
Приложение В	89
(обязательное)	89

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

БД – База данных

ПИ – Пользовательский интерфейс

ПЗ – Пояснительная записка

ПО – Программное обеспечение

СУБД – Система управления базами данных

MVC – Паттерн проектирования Модель, Вид, Контроллер

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Передача информации между двумя и более хостами в сети интернет в настоящий момент осуществляется множеством способов, в частности с помощью приложений WhatsApp, Viber, а также с помощью телефонной связи (SIP-телефония).

В любом из вышеописанных способов существуют недостатки. Приложения вышеописанного формата требуют обновления и без них не могут продолжать корректное функционирование. SIP-телефония требует наличия аппаратных и/или программных телефонных аппаратов, серверного программного обеспечения, что с технической точки зрения делает данный вид передачи данных дорогим и зависимым от поставки аппаратуры и услуг.

В связи с вышеизложенным перспективным направлением является технология передачи данных в рамках технологии HTML5 по схеме «точка - точка» или «точка - сервер - точка» с использованием «тонких» клиентов и API облачных серверов.

В концепции гиперссылок HTML5 применяется технология Hypermedia (технология WebRTC), которая расширяет возможности обработки и передачи веб-контента по сети.

Эта технология позволяет синхронизировать, структурировать и передавать контент (содержимое веб-страниц), в который интегрированы голосовые сообщения, видео-звонки, информация в виде текста и графики по схеме «точка - точка» или «точка - сервер - точка».

Технология WebRTC обеспечивает обмен данными между веб-браузерами в режиме реального времени без необходимости установки дополнительного программного обеспечения. Архитектура приложений веб-браузера в настоящий момент поддерживает гипермедиа в виде встроенных в браузер модулей и API, что делает WebRTC технологией подходящей для реализации целей дипломного проекта.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

Современная платформа веб-браузера позволяет создавать приложения, работающие независимо от операционной системы, с возможностью интегрировать библиотеки и API необходимые для работы с видео информацией.

Целью дипломного проекта является разработка приложения для передачи информации по схеме «точка -сервер- точка», ориентированного на веб-платформу с наличием следующих технологических решений:

- встроенная поддержка WebRTC;
- возможность связи по гиперссылкам между хостами одноранговой сети;
- поддержка различных типов мультимедиа, текстовых редакторов, обмен мгновенными сообщениями в текстовом виде.

Используемый стек технологий для реализации приложения:

- кроссплатформенный браузер;
- облачный сервер, с встроенной поддержкой API;
- операционная система Linux;
- язык программирования NodeJS.

Проект реализуется по схеме - минимально жизнеспособный продукт, обладающий минимальными, но достаточными для удовлетворения первых потребителей функциями. Основная задача проекта - получение обратной связи заказчика для формирования гипотез дальнейшего развития продукта.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1. Предпроектный анализ предметной области

1.1 Анализ предметной области

Программное обеспечения для передачи данных, позволяющие пользователям общаться, в дополнение к текстовой переписке, в аудио и видео режиме являются частью корпоративной культуры РФ.

Практически все приложения поддерживающие данные функции платные, зарубежные и с закрытым исходным кодом: whatsapp, viber, telegram, zoom и прочие.

Что является проблемой для корпоративного и государственного сектора, который хочет иметь в пользовании собственную программную платформу для проведения конференция, которая является:

- бесплатным или условно бесплатным программным продуктом;
- имеет открытый программный код;
- имеет документацию и техническую поддержку с возможностью самостоятельной доработки программного обеспечения для нужд заказчика.

В связи с выше озвученными проблемами, разработка и использование свободно распространяемого ПО с открытым исходным кодом является актуальным вопросом для любой крупной организации. Так как сотрудники для выполнения своих трудовых задач должны общаться с помощью средств связи:

- передавать друг другу медиа данные;
- устраивать конференции и брифинги в режиме онлайн.

Для реализации подобных приложений существуют способы, основанные на протоколах передачи данных: HTTP, WebSocket, gRPC или WebRTC.

В настоящее время аналогами западного программного обеспечения передачи потоковых данных являются следующие отечественные

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

программные системы: Контур.Толк, TrueConf Online, Яндекс.Телемост, Видеозвонки.Mail.ru.

В таблице 1.1. представлен сравнительный анализ данных систем, с указанием достоинств и недостатков.

Таблица 1.1 – Сравнительный анализ ПО (плюсы и минусы)

Возможности	Контур. Толк	TrueConf	Яндекс. Телемост	Видеозвонки. Mail.ru
Запись разговоров	имеется	имеется		
Работа с разных устройств	да	да		
Поиск	да	нет	да	да
Текстовый чат	нет	нет	да	да
Поддерживаемые Платформы	Windows Linux, Android Mac IOS WEB	Windows Android Mac IOS WEB	Windows, Linux, Android IOS WEB	Android IOS WEB
Развёртывание	облачное	облачное, локальное	облачное	облачное
Открытый код	нет	нет	нет	нет
Бесплатное пользование	нет	нет	да	да

Как показал сравнительный анализ, большинство современного программного обеспечения передачи потоковых данных являются:

- облачными решениями, которые не позволяют развернуть серверную часть приложения на аппаратной платформе заказчика и полностью ее контролировать;

- закрытыми системами с отсутствием доступа к исходным кодам;
- не все реализованное программное обеспечение кроссплатформенное.

Из вышесказанного и анализа приведенного в таблице 1.1 было принято решение реализовать программное обеспечение передачи потоковых данных реализующее следующие функции:

- создание сессии видео связи с поддержкой звука;
- остановка сессии видео связи;
- генерирование трафика видео связи;
- проверка канала связи, путем создания сессии от источника и обратно к тому же источнику без специальной обработки или модификаций (loopback);
- создание сессии в режиме конференция (инициатор и клиенты);
- вся инфраструктура будет размещена, развернута и использоваться в рамках учреждения заказчика, с полным контролем структурного подразделения по обслуживанию информационных технологий и информационной безопасности.

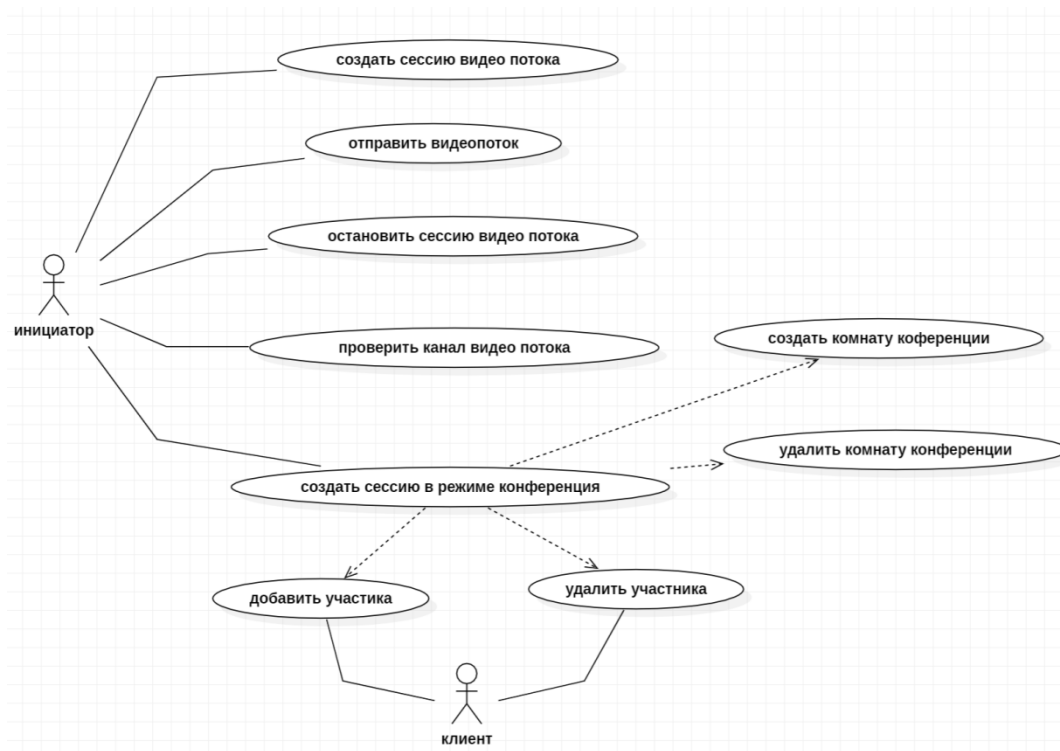


Рисунок 1.1 – Диаграмма вариантов использования.

Вышесказанное указывает на актуальность задачи реализации программного обеспечения передачи потоковых данных.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

1.2 Техническое задание

1.2.1 Назначение разработки

Программное обеспечение передачи потоковых данных предназначено для организации конференций и индивидуальных онлайн сессий в рамках вычислительной сети учреждения.

Программное обеспечение позволяет проводить совещания, брифинги, митапы, индивидуальные трудовые переговоры на базе платформы полностью контролируемой организацией заказчиком.

Каждый сотрудник предприятия, зарегистрированный службой информационной поддержки на сервере передачи потоковых данных имеет возможность использовать данное ПО для реализации своих служебных обязанностей.

1.2.2 Требования к программе

В связи с тем, что требования к разрабатываемому ПО делятся на категории, ниже представлены разделы которые описывают предъявленные к ПО требования.

1.2.2.1 Требования к функциональным характеристикам

Клиентская часть программного обеспечения передачи потоковых данных должна обеспечивать выполнение следующих функций:

- создать и послать серверу команду на создание сессии видео связи с поддержкой звука;
- создать и послать серверу команду на остановку сессии видео связи;
- принимать и визуализировать трафик видео связи от сервера;
- создать и послать серверу команду проверки канала связи;
- создать и послать серверу команды для работы в режиме конференция (инициатор и клиенты).

Входными данными являются:

- команды клиента для сервера передачи потоковых данных;
- команды клиенту от сервера передачи потоковых данных;

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						12
Изм.	Лист	№ докум.	Подпись	Дата		

- поток видео и аудио информации от клиента серверу;
- поток видео и аудио информации от сервера клиенту.

1.2.2.2 Требования к составу и параметрам технических средств

Система предназначена для работы на x86-совместимых персональных компьютерах со следующей минимальной конфигурацией: – процессор Intel Core 2.4 ГГц и выше, оперативная память 8 Гбайт и выше; жесткий диск; монитор.

1.2.2.3 Требования к информационной и программной совместимости

Система представляет собой клиент серверную платформу учреждения на базе, которого располагается серверная часть системы.

Серверная часть разворачивается на серверном кластере в составе серверного и коммутационного оборудования совместимого для работы с системами виртуализации и медиа сервером Kurento.

1.3 Стадии и этапы разработки

Стадии и этапы разработки программного обеспечения передачи потоковых данных представлены в таблице 1.2.

Таблица 1.2 - Стадии и этапы разработки ПО

Наименование этапов работы	Объем работы	Срок выполнения
Анализ предметной области	8%	
Составление технического задания	6%	
Выбор инструментов разработки	8%	
Проектирование системы	12%	
Развертывание и настройка медиа сервера	25%	
Разработка программы	25%	
Выполнение задания по БЖД	8%	
Расчет экономической целесообразности проекта	4%	

Наименование этапов работы	Объем работы	Срок выполнения
Оформление пояснительной записки и иной документации	4%	

1.4 Анализ стека технологий

Для реализации программного обеспечения передачи потоковых данных существует способы, основанные на следующих протоколах передачи данных: HTTP, WebSocket, gRPC или WebRTC.

В данном разделе проведем анализ протоколов, который покажет преимущества выбранного подхода для реализации дипломного проекта.

1.4.1 Протокол HTTP

Протокол HTTP - это прикладной протокол для распределенных информационных систем совместной работы и гипермедиа. HTTP –является универсальным протоколом для большинства клиент-серверных приложений обмена данными в сети интернет (рисунок 1.2).

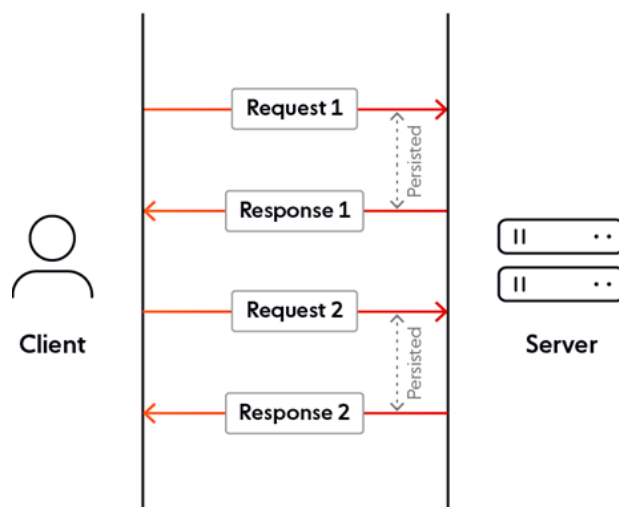


Рисунок 1.2. – Протокол передачи данных HTTP

Достоинства протокола HTTP:

- все браузеры поддерживают протокол HTTP и HTTPS с установкой SSL-сертификата;

- HTTP позволяет клиенту отправлять все запросы одновременно по одному TCP-соединению. TCP - это надежный, стабильный способ соединения.

Недостатки протокола HTTP:

- одновременные запросы могут увеличить нагрузку на серверы. Серверы HTTP могут получать запросы большими пакетами, что может привести к истечению времени ожидания запросов;
- проблему резкого увеличения нагрузки на сервер можно решить, установив прокси-сервер, который приведет к ограничению трафика.

1.4.2 Протокол WebSocket

Протокол WebSocket позволяют серверу и клиенту отправлять сообщения без связи с предыдущим запросом (рисунок 1.3).

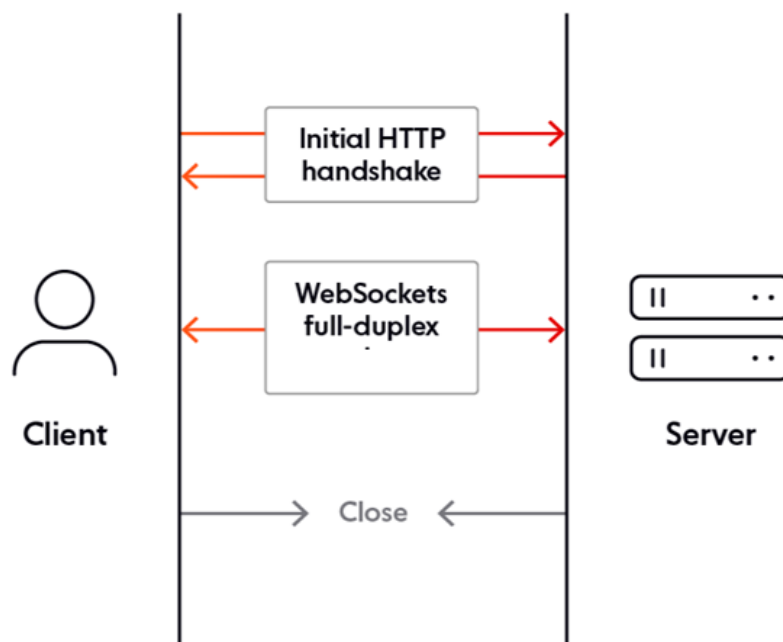


Рисунок 1.3. – Протокол передачи данных WebSocket

Все браузеры поддерживают протокол WebSocket без установки SSL-сертификата.

Достоинства протокола WebSocket:

- клиент и сервер могут инициировать сессию;
- клиент и сервер могут одновременно разговаривать друг с другом.

Недостатки протокола WebSocket:

- сессия не восстанавливается автоматически при прерывании соединения;
- браузеры старше 2015 года не поддерживают подключения по протоколу WebSocket.

1.4.3 Протокол gRPC

Протокол gRPC поддерживает дуплексную связь и позволяет серверу отправлять сообщения клиенту без опроса со стороны клиента.

Клиентская программа gRPC (рисунок 1.4) может вызывать процедуры или функции, размещенные на удаленном сервере, так же, как если бы они были реализованы локально в библиотеке кода.

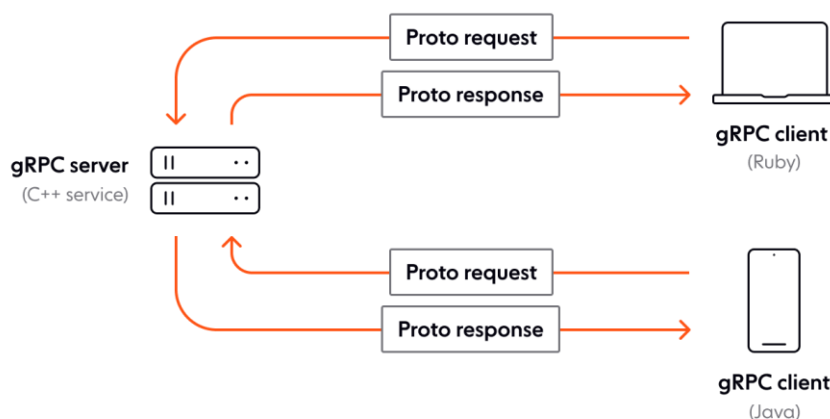


Рисунок 1.4. – Протокол передачи данных gRPC

Достоинства протокола gRPC:

- наличие большого количества инструментов и библиотек с открытым исходным кодом для разработки ПО.

Недостатки протокола gRPC:

- удаленные процедуры недоступны для чтения программистом, что затрудняет анализ программы на этапе отладки;
- протокол gRPC имеет проблемы с задержкой, так как постоянно происходит обмен служебной информацией между сетевыми частями приложения.

1.4.4 Протокол и платформа WebRTC

Протокол WebRTC - проект с открытым исходным кодом, который обеспечивает возможности голосовой, текстовой и видеосвязи в реальном времени между веб-браузерами и устройствами.

WebRTC предоставляет разработчикам программного обеспечения интерфейсы прикладного программирования (API), написанные на JavaScript с поддержкой различных языков программирования.

Разработчики используют WebRTC API (рисунок 1.5) для создания одноранговой соединения между браузерами и приложениями. Совместимость и поддержка аудио, видео и текстовых сообщений достигается встроенными средствами протокола. При использовании программных платформ поддерживающих WebRTC API: Janus, Kurento, совместимость и кроссплатформенность достигается встроенными в платформы библиотеками.

Классическая архитектура WebRTC приложения представлена на рисунке 1.5. Для работы приложения необходимо наличие сигнального и медиа серверов.

Сигнальный сервер осуществляет «рукопожатие» между двумя устройствами, на которых открыт браузер с поддержкой протокола WebRTC. После того как два устройства удостоверятся в доверительном соединении между собой сигнальный сервер передает управление устройствам и медиа серверу, которые начинают передавать медиа трафик по схеме: «точка - точка» или «точка-сервер-точка».

Если по какой-либо причине устройства не могут передавать данные, они обращаются к сигнальному серверу, который проверяет канал связи и создает его для передачи данных.

Достоинства протокола WebRTC:

- наличие плагинов и механизмов регулировать качество, полосу пропускания и поток трафика;
- поддерживается большинством веб-браузеров;
- кроссплатформенность относительно операционной системы;

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						17
Изм.	Лист	№ докум.	Подпись	Дата		

- является бесплатным программным обеспечением с открытым исходным кодом.

Недостатки протокола WebRTC:

- нет встроенных механизмов обеспечения безопасности трафика;
- каждый пользователь должен установить соединение в режиме P2P, что создает проблему с пропускной способностью;
- большие затраты на обслуживание классического соединения.

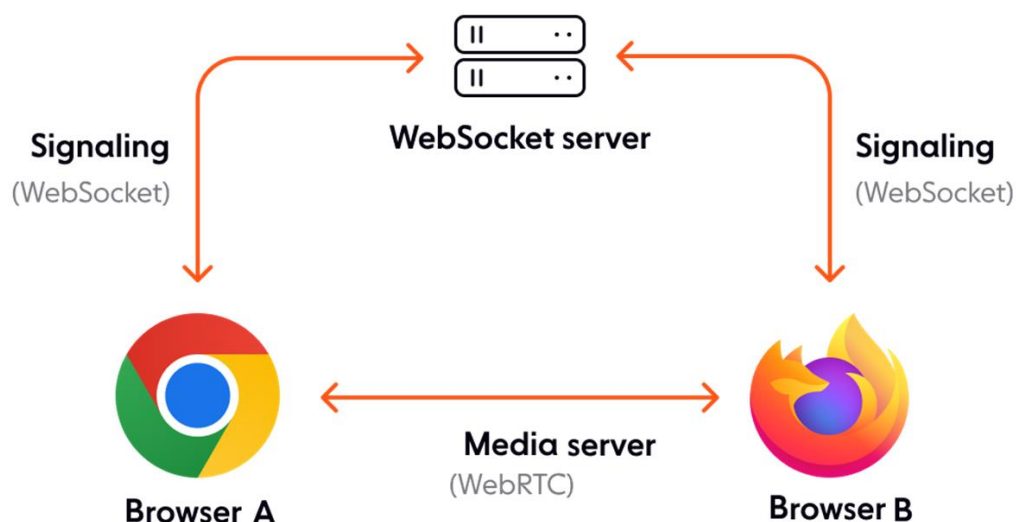


Рисунок 1.5 – Протокол передачи данных WebRTC.

В связи с тем, что протокол WebRTC лишен недостатков своих конкурентов, в дипломном проекте он будет использован в качестве протокола связи.

1.5 Выбор платформы медиа сервера для реализации проекта

Основная проблема передача данных через WebRTC это нагрузка на клиентов и сеть при большом количестве участников (рисунок 1.6). Количество соединений между участниками растёт в геометрической прогрессии, при этом ухудшается качество видео и растёт нагрузка на трафик и ресурсы системы.

Программные приложения на базе протокола WebRTC можно использовать как обычную P2P коммуникацию между 2-6 пользователями[4].

При большем количестве соединений нужен медиасервер, который контролирует полосы пропускания каждой сессии [5]. Также в классической реализации ПО появляются сложности, если требуется сохранять запись переговоров, поскольку вся работа по обработке аудио и видео потока производится на стороне клиента.

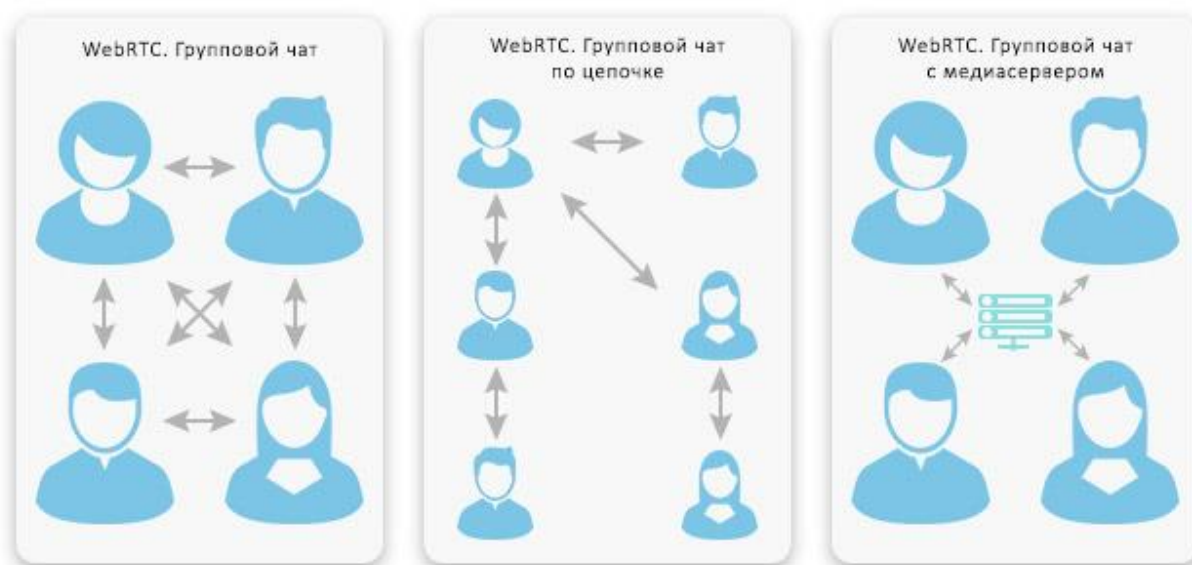


Рисунок 1.6 – Способы организации связи в программном обеспечении на базе протокола WebRTC.

Медиасервер необходим при реализации WebRTC приложения для работы с видео, аудио трафиком. Медиасервер как самостоятельное а не облачное решение необходимо, если у заказчика существует потребность в собственной инфраструктуре для коммуникаций, которая должна удовлетворять следующим потребностям:

- использует только бесплатные библиотеки или API для реализуемого приложения WebRTC, с открытым кодом;
- является автономной и независимой инфраструктурой для реализации WebRTC приложения;

В таблице 1.3 представлен сравнительный анализ медиасерверных платформ удовлетворяющих данным требованиям.

Таблица 1.3. –Сравнительный анализ медиасерверов WebRTC

Сервер	Режим работы	API	JS библиотеки	SDK	Запись в хранилище
Jitsi	SFU	Да	Да	Да	Да
Janus	SFU	Нет	Нет	Да	Да
Kurento	SFU/MCU	Да	Да	Да	Да
Mediasoup	SFU	Нет	Нет	Да	Нет
Medooze	SFU/MCU	Нет	Нет	Да	Да

Наличие режимов SFU (видеоконференция) и MCU (каждый участник подписывается только на один поток) позволяют реализовать конференц-связь (рисунок 1.7) и режим приватного разговора (рисунок 1.8). В связи с вышесказанным лучшим выбором для развертывания и последующей реализации программного обеспечения передачи потоковых данных является медиасервер Kurento.

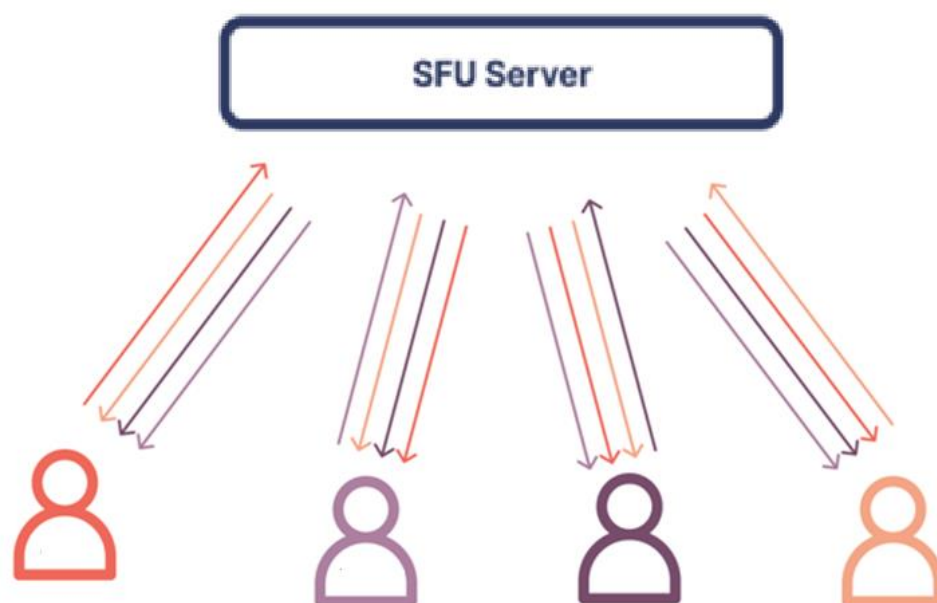


Рисунок 1.7 – Режим видеоконференция.

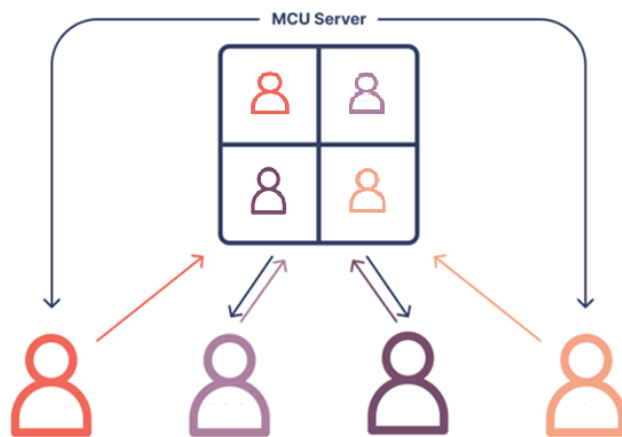


Рисунок 1.8 – Режим частных переговоров.

Медиасервер Kurento бесплатный WebRTC серверс открытым исходным кодом, который состоит из подключаемых при необходимости модулей (рисунок 1.9), обрабатывающих медиа потоки. Медиасервер включает встроенное API для разработки клиентских приложений и сервисов.

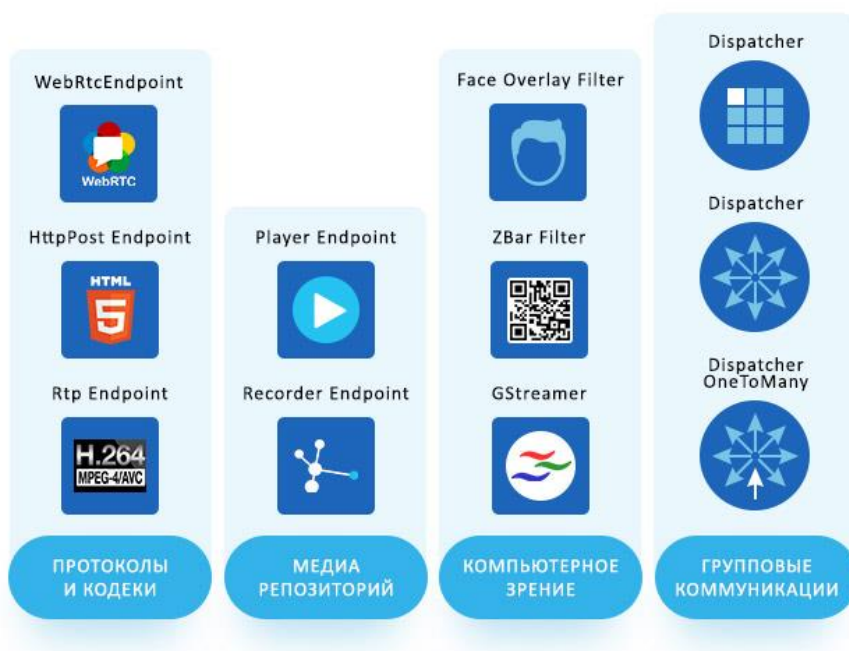


Рисунок 1.9.- Модули медиасервера Kurento.

Архитектура сервера Kurento состоит из медиа элементов и программных блоков, каждый из которых выполняет определенную функцию. Программный доступ к функциям медиа элементов предоставляется через встроенные функции API.

1.6 Выбор языка программирования

Для реализации программного обеспечения и связующего модуля со стороны сервера приложений требуется выбрать язык программирования.

Так как API сервера Kurento поддерживают языки программирования Java, Python, Node.js, приведем сравнительный анализ данных инструментов разработки в таблице 1.4.

Таблица 1.4. –Сравнение характеристик языков программирования

Характеристика	Java	Python	Node.js
Скорость исполнения	высокая	низкая	высокая
механизмы масштабирования модулей Kurento	имеются	нет	имеются
простота реализации однотипных проектов	сложно	просто	среднее
наличие стандартных библиотек	да	да	да
стоимость стандартных библиотек	имеются платные	полностью бесплатное	полностью бесплатное
кроссплатформенность	да	да	да

В связи с тем, что Node.js и язык полностью адаптированы для реализации взаимодействия с модулями медиасервера Kurento , выбор сделан в пользу Node.js.

1.7 Выбор системы управления базой данных

Платформа медиасервера Kurento поддерживает реляционные базы данных MySQL, PostgreSQL и MariaDB.

СУБД MySQL - простая в использовании база данных, которая подходит для простых задач и веб-приложений. СУБД MySQL имеет высокую производительность и скорость обработки запросов, которые делают ее популярной среди разработчиков и использования в WEBпроектах. MySQL подходит для неспецифических приложений с небольшими затратами на аппаратное обеспечение.

СУБД PostgreSQL- база данных предлагает мощный и масштабируемый фреймворк, который может обрабатывать высокие нагрузки. СУБД имеет широкий набор инструментов и поддерживает широкий диапазон типов данных. СУБД PostgreSQL используется в крупных корпоративных средах и научных проектах.

СУБД MariaDB - это одна из самых быстрых баз данных, доступных на рынке с высокой скоростью ответа и производительностью. В отличие от MySQL, MariaDB имеет современную архитектуру и более широкие возможности, спроектированные с учетом недостатков конкурентов. СУБД MariaDB подходит для крупных транзакционных систем и приложений с высокой доступностью в частности для реализации медиа серверов и систем передачи потоковых данных.

В данном проекте используется СУБД MariaDB, так как данная СУБД имеет функционал для интеграции с медиа серверами.

1.8 Выводы по первой главе

В результате первой главы проведен анализ предметной области, поставлена задача для реализации программного приложения и выбран стек технологий.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

2. Проектирование и реализация программного обеспечения передачи потоковых данных.

Архитектура реализованного приложения представлена на рисунке 2.1

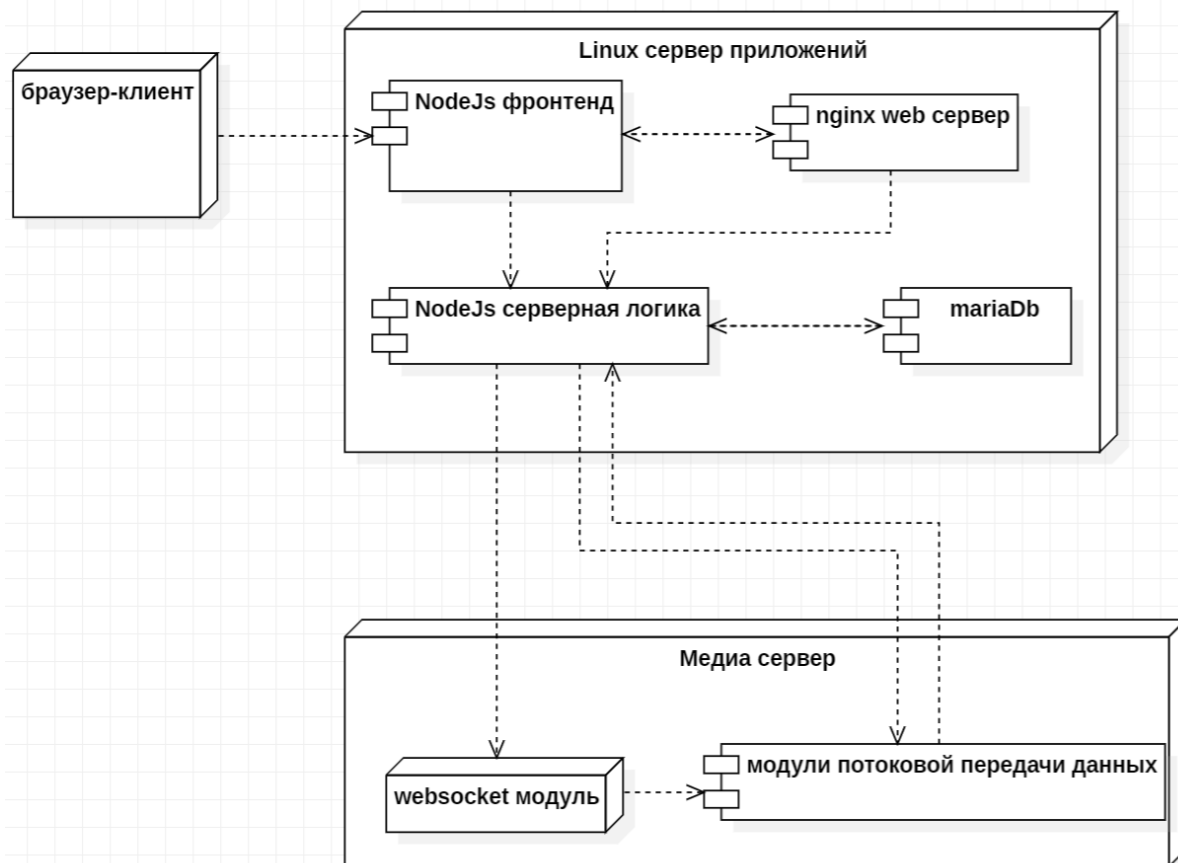


Рисунок 2.1 – Диаграмма развертывания

Как видно из рисунка 2.1 – это классическая трехзвенная архитектура аппаратно-программной платформы WebRTC.

Проектирование программного обеспечения передачи потоковых данных состоит из трех этапов:

- проектирование и реализация основной логики приложения;
- проектирование и реализация базы данных;
- макетирование и верстка интерфейсов пользователя с учетом представления СУБД.

2.1 Проектирование и реализация серверной части приложения

Медиа сервер Kurento работает с запросами от клиентской части приложения. В связи с тем, что медиа сервер состоит из стандартного набора модулей, существует следующий лимитированный набор команд, для создания, управления и закрытия потока данных:

- ping – команда проверки работоспособности сервера;
- create - создает новый объект, для работы с сервером;
- describe – показывает информацию о существующем объекте;
- invoke - вызывает функцию указанного модуля сервера;
- subscribe – указывает на событие, для получения уведомления;
- unsubscribe - удаляет указатель на событие;
- release – аналог delete, освобождает память сервера от удаленных ресурсов;
- connect - организует соединение с медиа-сервером,
- start–команда клиент о начале сессии.

На основании данных команд покажем схему взаимодействия частей приложения диаграммами последовательности (рисунки 2.1, 2.2).

2.1.1 Режим работы «проверка канала связи»

На рисунке 2.1 показана схема создания потока данных в режиме проверки канала связи, путем создания сессии от источника и обратно к тому же источнику без специальной обработки или модификаций.

Как видно из диаграммы последовательности на рисунке 2.2 клиентское приложение создает объект *WebRtcPeer*, с помощью команды *create(WebRtcPeer)*, необходимый для связи клиента с определенным потоком данных, который будет создан медиа сервером. Этот же объект *WebRtcPeer* необходим для работы с командами серверу приложений (сигнальному серверу).

Как только объект *WebRtcPeer* создан, клиентское приложение посылает команду серверу приложений начать сессию *connect(WebRtcPeer, loopback)*.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

Получив команду *connect* с признаком *loopback* сервер приложений дает команду медиа серверу *create(WebRtcPeer, loopback)* создать связь между входным потоком и выходным потоком модуля *WebRtcEndPoint* сервера Kurento. Модуль *WebRtcEndPoint* отвечает за создание потока данных с возможностью передачи видео и аудио информации.

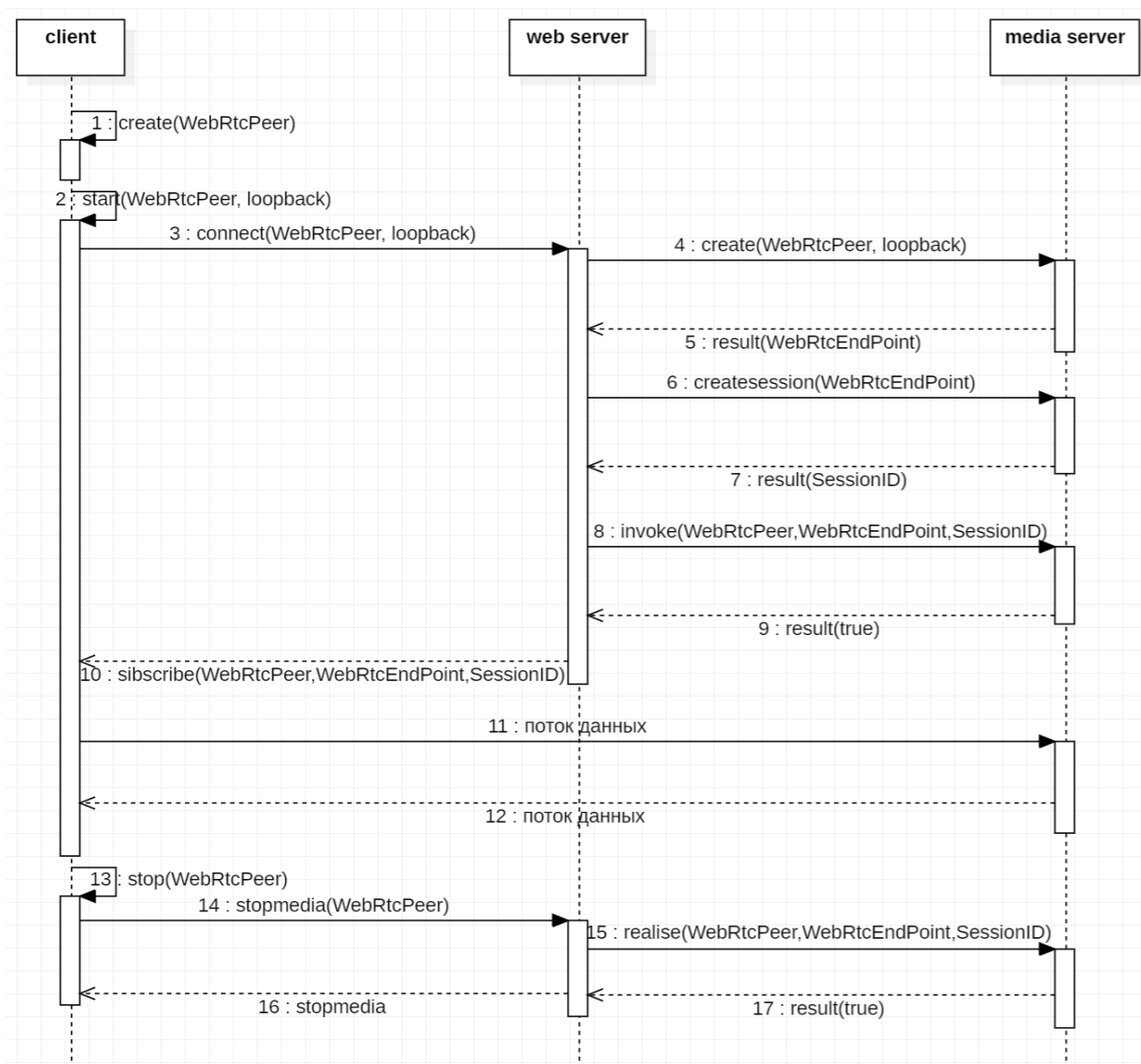


Рисунок 2.2 – Диаграмма последовательности в режиме проверки канала связи

В результате создания модуля *WebRtcEndPoint*, медиа сервер посылает серверу приложений ответ с экземпляром модуля *WebRtcEndPoint* и номером созданной сессии.

Далее происходит привязка клиента к конкретному модулю `WebRtcEndPoint`, которая позволит транслировать медиа поток через сервер к клиенту `sibscribe`, где `WebRtcPeer` – объект, указывающий на клиента, `WebRtcEndPoint` – объект, указывающий на модуль работы с потоком данных на стороне медиа сервера, `SessionID` – номер сессии, необходим для идентификации потока данных, так как один и тот же клиент и сервер могут одновременно работать с несколькими сессиями передачи данных.

Функции `invoke` и `sibscribe` создают связку «клиент-медиа сервер- объект `WebRtcEndPoint`». Как только данная связка образовалась, потоковые данные от клиента будут отправлены на медиа сервер и возвращены ему с помощью модуля `WebRtcEndPoint`.

Команда стоп на стороне клиента позволяет остановить поток данных и удалить информацию о текущей сессии.

2.1.2 Режим работы «конференция»

На рисунке 2.3 показана схема создания потока данных в режиме конференции, путем создания сессии, в которой присутствует клиент инициатор и клиент гость. Оба участника могут передавать друг другу видео и аудио информацию.

Клиентское приложение А создает объект `WebRtcPeerA`, с помощью команды `create (WebRtcPeerA)`, необходимый для связи клиента с потоком данных, который будет создан медиа сервером. Этот же объект `WebRtcPeerA` необходим для работы с командами серверу приложений (сигнальному серверу).

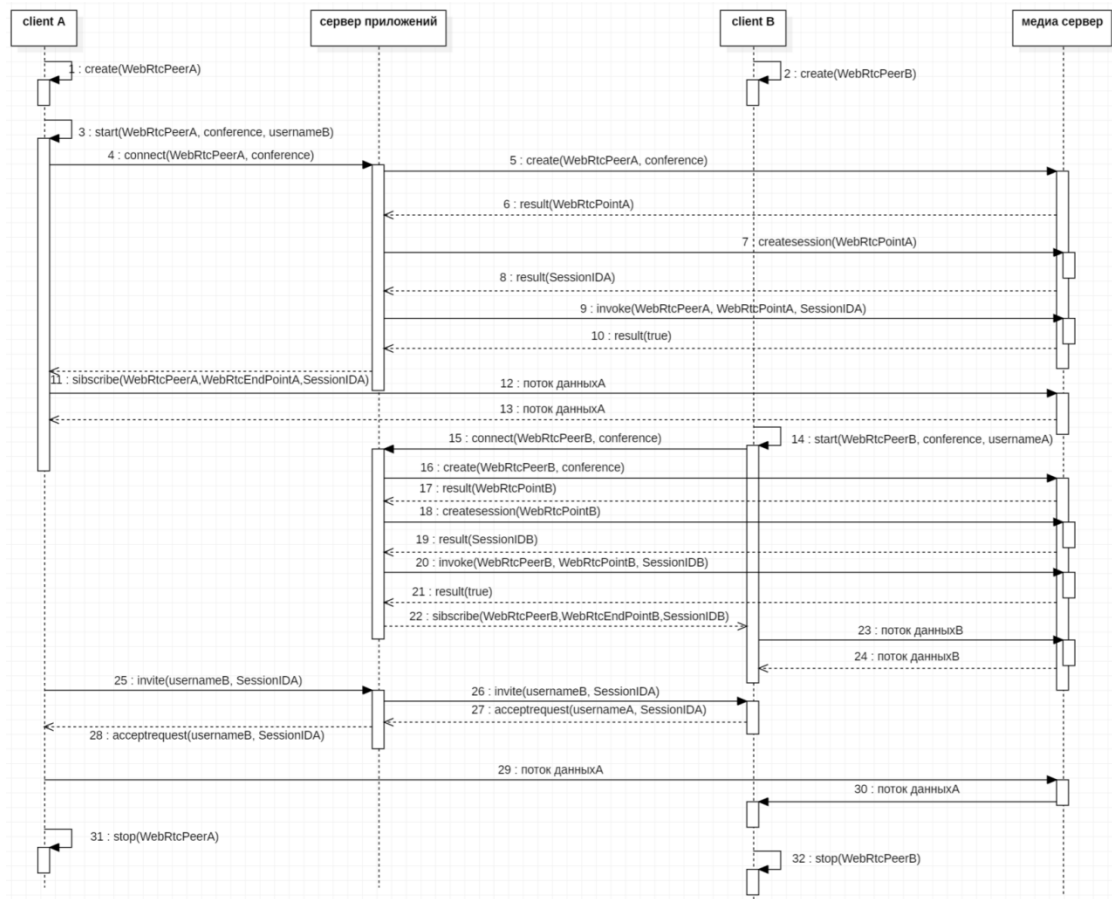


Рисунок 2.3 – Диаграмма последовательности в режиме конференции.

Объект *WebRtcPeerA* позволяет клиентскому приложению отправить команду серверу приложений для начала новой сессии *connect(WebRtcPeerA, conference)*. Получив команду *connect* с признаком *conference* сервер приложений отправляет команду медиа серверу *create(WebRtcPeerA, conference)* создать с модуль *WebRtcEndPointA* сервера Kurento. Модуль *WebRtcEndPointA* отвечает за создание потока данных с возможностью передачи видео и аудио информации.

Далее происходит привязка клиента А к модулю *WebRtcEndPoint*, что позволит транслировать поток данных через сервер к клиенту командой *sibscribe*, где *WebRtcPeerA* – объект, указывающий на клиента, *WebRtcEndPointA* – объект, указывающий на модуль работы с потоком данных на стороне медиа сервера, *SessionIDA* – номер сессии, необходим для идентификации потока данных, так как один и тот же клиент и сервер могут одновременно работать с несколькими сессиями передачи данных.

Как только выше перечисленные команды успешно будут выполнены клиент А получит видео поток в режиме loopbackи будет в нем находиться до тех пор пока к конференции не присоединиться первый клиент.

На рисунке 2.3 показано, что для клиента Впоследовательность действий таже что и для клиента А.

Далее клиент А с помощью команды *invite*, с указанием имени гостевого клиента и своей сессии приглашает клиента Вв сою конференцию.

Если клиент В на приглашение через сервер приложенийотправляет клиенту А команду *acceptrequest*то два клиента организуют единую сессию с единым медиа потоком.

Окончанием единой сессии считается получение сервером приложений команды *stop*от одного из клиентов.

После ее получения сервер приложений разрывает единый медиа поток и клиенты работают в режиме loopback.

Команды по цепочке: клиент, сервер приложений, медиа сервер передаются в формате JSON. Ниже представлен список и формат команд используемых в реализованном приложении:

- проверка работоспособности сервера;

```
{
  "id": 1,
  "method": "ping",
  "params": {
    "interval": 240000
  }
}
```

- создание нового объекта, для работы с сервером;

```
{
  "id": 2,
  "method": "create",
  "params": {
    "type": "MediaRtcPeer",
    "constructorParams": {},
    "properties": {conference}
  }
}
```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						29
Изм.	Лист	№ докум.	Подпись	Дата		

- запрос информации о существующем объекте;

```
{
  "id": 4,
  "method": "describe",
  "params": {
    "object": "ID_kurento.MediaPipeline",
    "sessionId": "0cd20d0e-451f-4fd9"
  }
}
```

- вызов требуемого модуля медиа сервера;

```
{
  "id": 6,
  "method": "invoke",
  "params": {
    "object": "ID_kurento[0].MediaPipeline/ WebRtcEndpoint",
    "operation": "connect",
    "operationParams": {
      "sink": "ID_kurento[1].MediaPipeline/ WebRtcEndpoint"
    },
    "sessionId": "bd4d6227-0463"
  }
}
```

- указание на событие, для получения уведомления;

```
{
  "id": 8,
  "method": "subscribe",
  "params": {
    "type": "StopStream",
    "object": "ID_kurento.MediaPipeline/WebRtcEndpoint",
    "sessionId": "bd4d6227-0463"
  }
}
```

- удаление указателя на событие;

```
{
  "id": 9,
  "method": "unsubscribe",
  "params": {
    "subscription": "052061c1-0d87-4fbd",
    "object": "ID_kurento.MediaPipeline/WebRtcEndpoint",
    "sessionId": "bd4d6227-0463-4d52"
  }
}
```

- команда, освободить память сервера от удаленных ресурсов;

```
{
  "id": 7,
  "method": "release",
  "params": {
    "object": "ID_kurento.MediaPipeline",
    "sessionId": "bd4d6227-0463-4d52"
  }
}
```

- создание соединения с медиа-сервером;

```
{
  "id": 10,
  "method": "connect",
  "params": {
    "sessionId": "4f5255d5-5695-4e1c"
  }
}
```

2.2 Проектирование клиентской части (интерфейса)

Дизайн web приложения является важным этапом при создании приложения. При проектировании дизайна разработчик акцентирует внимание на структуру (рисунок 2.4) и внешний вид приложения (рисунки 2.5 -2.9).

Этап проектирования охватывает несколько различных аспектов, включая дизайн пользовательского интерфейса (UI), удобство использования (UX), создание контента и графический дизайн.

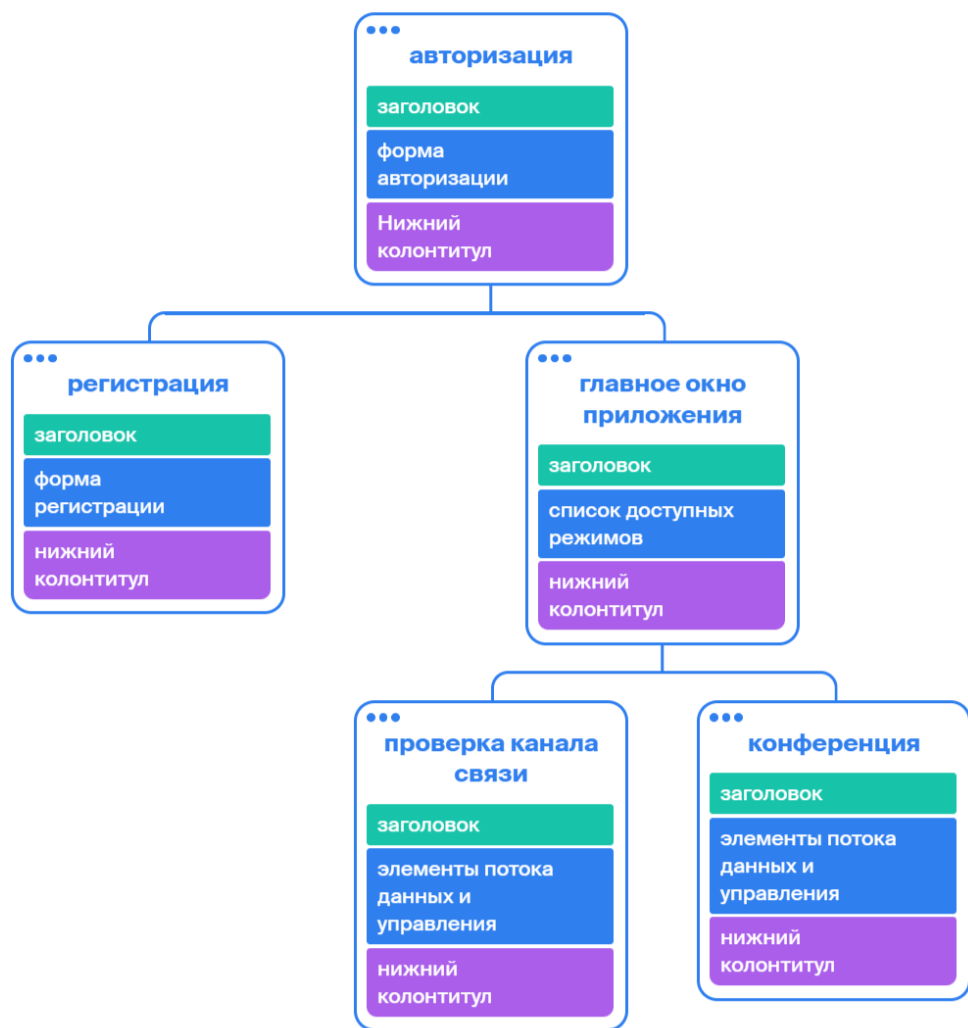


Рисунок 2.4 – Структура интерфейса приложения

Структура и функционал страниц приложения на рисунке 2.6. позволяют работать в режиме приема и передачи потоковых данных.

URL 192.168.0.25:8082

Программное обеспечение передачи потока данных

Вход

Логин

Пароль

Рисунок 2.5 – Страница доступа пользователя

Элементами страницы доступа пользователей являются объекты (рисунок 2.5.):

- форма авторизации – тип Form;
- поле ввода логина - тип Input type;
- поле ввода пароля - тип Input type;
- кнопка регистрации - тип Button;
- кнопка входа в систему тип Button.

Элементами страницы регистрации пользователей являются объекты (рисунок 2.6.):

- форма регистрации – тип Form;
- поле ввода имени - тип Input type;
- поле ввода фамилии - тип Input type;
- поле ввода отчества - тип Input type;
- поле ввода логина - тип Input type;
- поле ввода пароля - тип Input type;
- поле ввода подтверждение пароля - тип Input type;
- кнопка регистрации - тип Button;

- кнопка сброса - тип Button.

Рисунок 2.6 – Страница регистрации пользователей

Основная страница приложения представлена на рисунке 2.7.

Рисунок 2.7 – Основная страница приложения

Элементами страницы регистрации пользователей являются объекты (рисунок 2.7.):

- форма основной страницы – тип Form;
- поле отображения логина пользователя - тип href;
- кнопка выбора режима проверки канала связи - тип Button;
- кнопка выбора режима конференции - тип Button.

Страница режима проверки связи представлена на рисунке 2.8.

Рисунок 2.8 – Страница режима проверки связи

Элементами страницы регистрации пользователей являются объекты (рисунок 2.8.):

- форма основной страницы – тип Form;
- поле отображения логина пользователя - тип href;
- поле исходного видео потока – тип video;
- поле видео потока от медиа сервера – тип video;
- уровень звука – тип progress;
- кнопка «Начать» - тип Button;

- кнопка «Завершить» - тип Button.

URL

192.168.0.25:8082

Программное обеспечение передачи потока данных

Пользователь: novinska

Назад

novinska



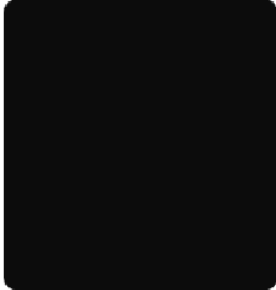
Начать

ivan



Начать

petrov



Начать

Рисунок 2.9 – Страница режима конференции

Элементами страницы регистрации пользователей являются объекты (рисунок 2.9.):

- форма основной страницы – тип Form;
- поле отображения логина пользователя - тип href;
- поле видео потока инициатора – тип video;
- поле видео потока от клиентов – тип video;
- кнопка «Начать» - тип Button;
- кнопка «Назад» - тип Button.

2.3 Проектирование базы данных приложения

Модель данных реализуемой БД состоит из четырех связанных между собой таблиц (рисунок 2.10): информация о персоне, информация об аккаунте, информация о сессии и информация логирования.

В таблице «Account» хранится информация о логине и пароле пользователя (в виде хеш свертки), а также информация о связях с таблицами персоны, сессии и таблицы логирования.

В таблице «Persona» хранится фамилия имя и отчество пользователей.

В таблице «Session» фиксируется время начало и окончания каждой сессии связанной с пользователем программы.

В таблице «Logs» хранятся пометки о пользователе и его сеансе.

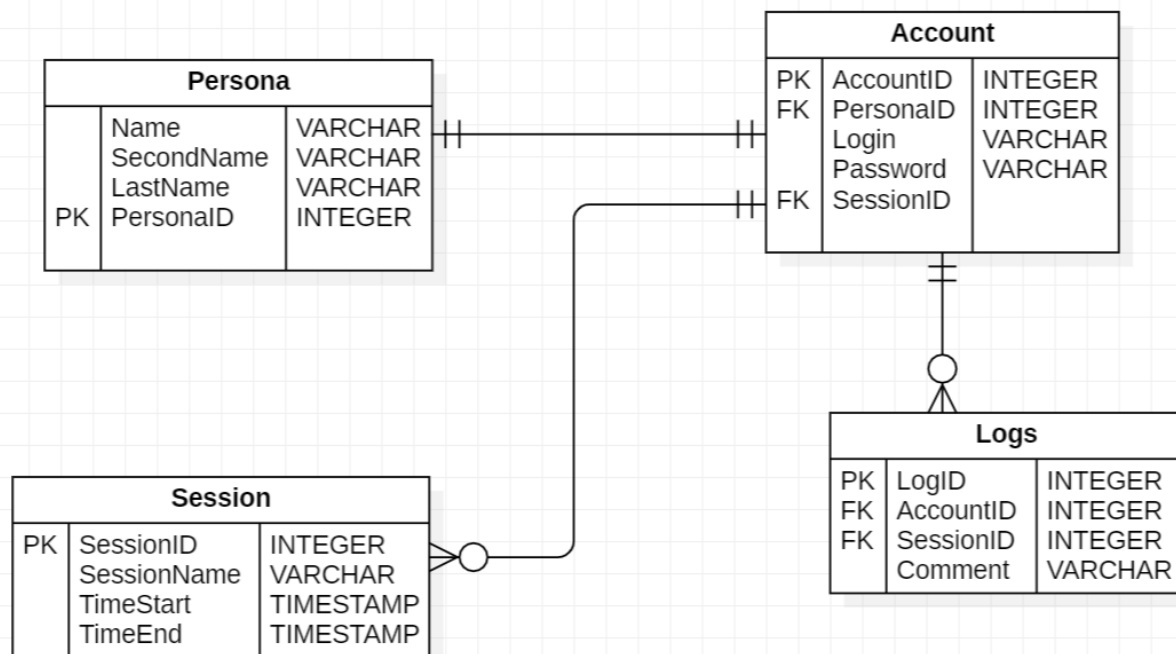


Рисунок 2.10 – Физическая структура базы данных приложения.

Таблица 2.1 – Описание полей таблицы «Persona»

Имя поля	Тип	Примечание
PersonID	int	Первичный ключ, обязательный
Name	varchar(128)	Обязательный
SecondName	varchar(128)	Необязательный
LastName	varchar(128)	Обязательный

Таблица 2.2 – Описание полей таблицы «Account»

Имя поля	Тип	Примечание
AccountID	int	Первичный ключ, обязательный
PersonID	int	Обязательный, внешний ключ

Имя поля	Тип	Примечание
Login	varchar(128)	Обязательный
Password	varchar(128)	Обязательный
SessionID	int	Обязательный, внешний ключ

Таблица 2.3 – Описание полей таблицы «Session»

Имя поля	Тип	Примечание
SessionID	int	Первичный ключ, обязательный
SessionName	varchar(128)	Обязательный
TimeStart	TimeStamp	Обязательный
TimeStop	TimeStamp	Обязательный

Таблица 2.4 – Описание полей таблицы «Logs»

Имя поля	Тип	Примечание
LogID	int	Первичный ключ, обязательный
AccountID	int	Обязательный
SessionID	int	Обязательный
Comment	varchar(128)	Обязательный

В результате проектирования базы данных созданы сущности и связи, которые позволят хранить информацию для запуска приложения.

2.4 Выводы по второй главе

В результате второй главы реализовано программное обеспечение передачи потоковых данных, для чего были выполнены следующие этапы:

- спроектирована и реализована основная логика приложения;
- реализована база данных приложения;
- реализован макет и верстка интерфейсов пользователя.

3. Технология развертывания сервера программного обеспечения передачи потоковых данных.

В данной главе продемонстрированы основные этапы настройки и запуска сервера для передачи потоковых данных на базе сервера Linux.

Для развертывания медиа сервера на базе ОС Linux скачайте и распакуйте образ с официального сайта производителя.

Существует два способа установки медиасервера Kurento в ОС Ubuntu Server:

- локальная установка на Ubuntu;
- установка с помощью образа Docker.

3.1 Локальная установка медиа сервера на Ubuntu

Прежде чем приступить к работе, рекомендуется обновить все доступные пакеты до их последних доступных версий:

```
sudo apt update
```

```
sudo apt upgrade
```

Кроме того, установите необходимые пакеты с помощью команды:

```
sudo apt install --no-install-recommends gnupg
```

Импортируйте подпись ключа GPG для репозитория:

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 5AFA7A83
```

Так как Kurento официально поддерживает **Ubuntu 18.04 (Bionic)**, требуется экспортировать данный дистрибутив ОС Ubuntu:

```
DISTRIB_CODENAME= «bionic»
```

Добавьте репозиторий в систему операционной среды:

```
sudo tee "/etc/apt/sources.list.d/kurento.list"
```

```
# Kurento Media Server
```

```
deb [arch=amd64] http://ubuntu.openvidu.io/ $DISTRIB_CODENAME kms6  
EOF
```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						39
Изм.	Лист	№ докум.	Подпись	Дата		

Как только репозиторий будет добавлен в ОС Linux, установите KMS с помощью команды:

```
sudo apt install libopencv-dev
sudo apt-get update ; sudo apt-get install
--no-install-recommends\kurento-media-server
```

После установки сервисом Kurento можно управлять с помощью команд:

```
sudo service kurento-media-server start
sudo service kurento-media-server stop
```

Запущенный сервер Kurento представлен на рисунке 3.1.

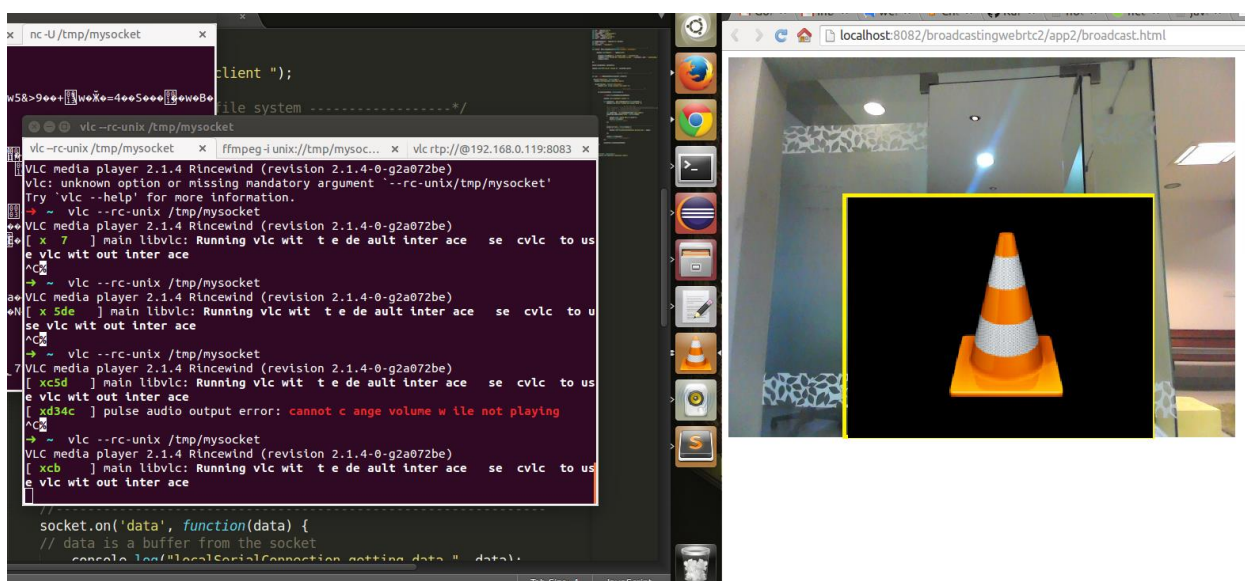


Рисунок 3.1 – Запущенный сервис Kurento

3.2 Установка медиа сервера с помощью образа Docker

Для начала установки с помощью Docker необходимо, чтобы он был установлен на сервере.

Для установки контейнеров docker выполните следующие команды:

```
curl -fsSL get.docker.com -o get-docker.sh
sudo sh get-docker.sh
```

Проверьте правильность установки.

```
$ docker --version
```

Docker version 20.10.17, build 100c701

Запустите и включите службу Docker

```
sudo systemctl enable --now docker
```

Добавьте пользователя вашей системы в группу docker.

```
sudo usermod -aG docker $USER
```

```
newgrp docker
```

После установки Docker запустите контейнер Kurento WebRTC со всеми открытыми портами.

```
docker run -d \  
  --name kurento \  
  -p 8888:8888/tcp \  
  -p 5000-5050:5000-5050/udp \  
  -e KMS_MIN_PORT=5000 \  
  -e KMS_MAX_PORT=5050 \  
  kurento/kurento-media-server:latest
```

Проверьте корректность запуска контейнера:

```
$ docker ps  
7e30ccd4befa kurento/kurento-media-server:latest "/entrypoint.sh"  
0.0.0.0:5000-5050->5000-5050/udp, :5000-5050->5000-5050/udp,  
0.0.0.0:8888->8888/tcp, : 8888->8888/tcp kurento
```

После установки медиа сервера требуется произвести его настройки. Сервер Kurento WebRTC хранит свои настройки в следующих файлах:

- /etc/kurento/kurento.conf.json- основной конфигурационный файл, который определяет поведение сервера;
- /etc/kurento/modules/kurento/MediaElement.conf.ini хранит информацию о модулях и функциях MediaElement;
- /etc/kurento/modules/kurento/SdpEndpoint.conf.ini -параметры аудио и видео для клиентов через модули WebRtcEndpoint и RtpEndpoint;
- /etc/kurento/modules/kurento/WebRtcEndpoint.conf.ini - параметры для WebRtcEndpoint.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						41
Изм.	Лист	№ докум.	Подпись	Дата		

Далее требуется установить дополнительные пакеты для запуска клиентского приложения:

```
curl -sL https://deb.nodesource.com/setup_16.x | sudo -E bash
```

```
sudo apt-get install -y nodejs
```

```
sudo npm install -g brower
```

После чего запустим команду для инсталляции клиентского приложения из репозитория:

```
git clone https://github.com/novinsky/appclient.git
```

```
cd appclient /appmedia
```

```
git checkout master
```

```
npminstall
```

Запустите сервер Kurento, установленный на локальном хостинге , с помощью команды:*npm start* .

Теперь просмотрите демонстрационную страницу в браузерах, совместимых с WebRTC, таких как Chrome и Firefox, используя URL (рисунки 3.2 и 3.3 соответственно).

Как видно на рисунке 3.3, потоковый трафик данных заработал, и на экране появилось медиа изображение. Данный факт подтверждает что настройка сервера и развертывание приложения произведено успешно.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						42
Изм.	Лист	№ докум.	Подпись	Дата		

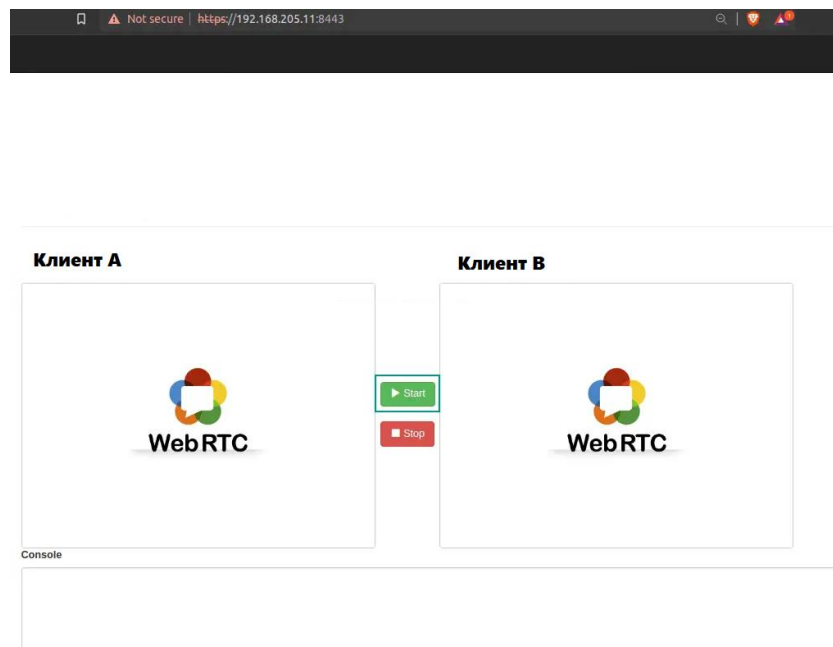


Рисунок 3.2 – Клиентская часть приложения до запуска

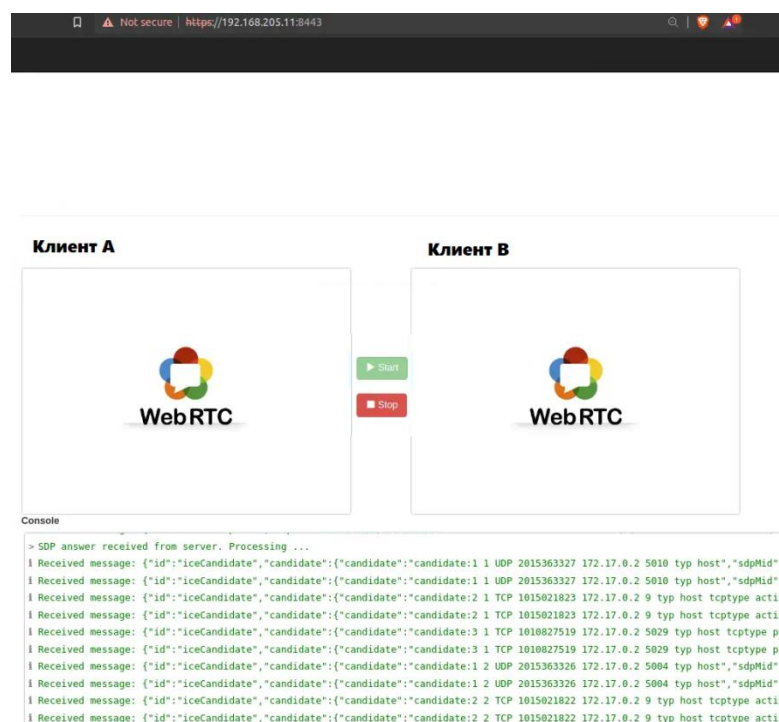


Рисунок 3.3 – Клиентская часть приложения после запуска

3.3 Выводы по третьей главе

В третьей главе показаны способы инсталляции медиа сервера и клиентского приложения для передачи потоковых данных.

4. Экономическое обоснование эффективности разработки и внедрения программы потоковой обработки данных

Разработка программного обеспечения осуществлена на базе собственной производственной площади общей площадью 160 кв.м. Содержание помещения в месяц составляет 15 000 рублей.

Для разработки программного обеспечения приобретена вычислительная техника, лицензионное программное обеспечение и расходные материалы (обучающие материалы).

Данные по потребности в оборудовании и лицензионном программном обеспечении представлены соответственно в таблице 4.1 и 4.2.

Таблица 4.1 – Потребность в оборудовании

Оборудование	Источник	Кол-во	Ориентировочная цена за единицу, руб	Потребляемая мощность, Ватт
Компьютер Lime 410	ООО«ДНС»	5	51000	136
Сервер Lime S510	ООО«ДНС»	1	172000	500
Клавиатура и мышь	ООО«ДНС»	5	900	2,5
Монитор НРН2718	ООО«ДНС»	5	15500	18

Таблица 4.2 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена, рублей	Срок действия лицензии, дней
Microsoft Windows 10 Pro	интернет-магазин ООО «1Бит»	29000	бессрочная
Антивирус Kaspersky 2023/2024	интернет-магазин ООО «1Бит»	1999	365

Перечень дополнительных материалов, приобретение которых не требуется, приведены в таблице 4.3.

Таблица 4.3 – Потребность в дополнительных материалах

Материал	Источник	Количество, штук
Стол рабочий с надстройкой	ООО «Мебельер»	1
Настольная лампа	ООО «Мебельер»	5
Стол компьютерный	ООО «Мебельер»	4
Кресло	ООО «Мебельер»	4
Канцтовары	ООО «Канцлер»	5
Бумага	ООО «Канцлер»	5

Разработка программного обеспечения осуществлялась командой разработчиков, состав которой представлен в таблице 10.4

Состав коллектива разработчиков и информация по заработной плате разработчиков программного обеспечения и их количественному составу приведены в таблице 4.4.

Таблица 4.4 – Занятость и заработная плата разработчиков

Специалист	Количество, человек	Сумма заработной платы, рублей
Руководитель проекта	1	90000
Системный архитектор	1	70000
Специалист по защите информации	1	90000
Программист	1	70000
Тестировщик	1	60000

Себестоимость разработки программного продукта включает следующие статьи затрат:

- затраты на приобретение оборудования;
- затраты на приобретение лицензионного программного обеспечения;
- затраты на дополнительные материалы;
- затраты на заработную плату;
- затраты на обслуживание.

Затраты на оборудование в период разработки составят 509000 рублей на основании данных в таблице 4.1.

Затраты на программное обеспечение в период разработки составят 154995 рублей и представлены в таблице 4.2.

Затраты на дополнительные материалы в период разработки составят 0 рублей в соответствии с условием что потребности в закупке нет.

Затраты на заработную плату включают затраты на основную и дополнительную заработную плату разработчиков программного обеспечения и отчисления на страховые взносы.

Затраты на основную заработную плату разработчиков определяются исходя из установленной заработной платы, количества специалистов и их занятости в процессе разработки программного продукта (продолжительность работы в днях).

Дополнительная заработная плата составляет 15% от затрат на основную заработную плату разработчиков.

С учетом выходных дней при 5-дневной рабочей неделе количество рабочих дней в месяце в среднем составляет 21 день.

Затраты на основную и дополнительную заработную плату разработчиков программного обеспечения представлены в таблице 4.5.

Таблица 4.5 – Затраты на основную и дополнительную заработную плату разработчиков

Исполнитель	Количество, человек	Сумма заработной платы, рублей	Период занятости, дней	Основная заработная плата, рублей	Дополнительная заработная плата, рублей	Всего рублей
Руководитель проекта	1	90000	63	270000	40500	310500
Системный Архитектор	1	70000	63	210000	31500	241500

Исполнитель	Количество, человек	Сумма заработной платы, рублей	Период занятости, дней	Основная заработная плата, рублей	Дополнительная заработная плата, рублей	Всего рублей
Специалист по защите информации	1	90000	63	270000	40500	310500
Программист	1	70000	63	210000	31500	241500
Тестировщик	1	60000	63	180000	27000	207000
Итого	5	380000	315	1140000	171000	1311000

Таким образом, затраты на заработную плату разработчиков составят 1138500 рублей.

Отчисления на страховые взносы по нормам, установленным законодательством в ПФР, ФОМС, ФСС РФ вычисляются исходя из данных представленных ниже.

Отчисления на страховые взносы производится согласно «Налоговому кодексу Российской Федерации (часть вторая)» от 05.08.2000 № 117-ФЗ (ред. от 26.03.2022).

Действующие тарифы на страховые взносы работодателей установлены статьей 34 НК РФ и составляют: ОПС (обязательное пенсионное страхование) – 22%, ВНиМ (страхование временной нетрудоспособности и в связи с материнством) – 2,9%, ОМС (обязательное медицинское страхование) – 5,1%.

Общая сумма отчислений на страховые взносы составит 393300 рублей.

Общая сумма затрат является суммой: затрат на оплату труда включая налоги и сборы, затраты на закупку оборудования и лицензионного ПО и составляет 1974995 рублей.

При этом учреждение оказывает техническую поддержку самостоятельно внутренним клиентам и самостоятельно реализует дополнительные модули клиентского приложения собственной командой разработки.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						49
Изм.	Лист	№ докум.	Подпись	Дата		

5. Безопасность жизнедеятельности. Особенности микроклимата на рабочем месте оборудованном ЭВМ.

Микроклимат помещения это состояние внутренней среды помещения, оказывающей непосредственное воздействие на организм человека.

Прежде чем судить о микроклимате помещения и принимать какие-то решения по его корректировке, нужно определенным образом и по определенным параметрам «измерить» его реальное состояние. Существуют санитарно-эпидемиологические правила и нормативы СанПиН 2.2.2/2.4, которые включают в себя следующий пункт:

- санитарные нормы дают четкую градацию условий микроклимата производственных помещений. В соответствии с этим документом условия окружающей среды подразделяются на оптимальные и допустимые.
- оптимальные микроклиматические условия отличаются тем, что они обеспечивают полный комфорт тепловому и функциональному состоянию организма человека в течение восьмичасовой рабочей смены. Происходит это при минимальном напряжении механизмов терморегуляции, не вызывает отклонений в состоянии здоровья. Оптимальные условия микроклимата создают предпосылки для высокого уровня работоспособности и являются предпочтительными на рабочих местах.

В обязательном порядке эти условия устанавливаются на рабочих местах помещений, на которых выполняются работы операторского типа. Об этом прямо сказано в пункте 5.2 Санитарных правил. Обычно эти работы связаны с нервно-эмоциональным напряжением человека (работа в кабинах, на пультах и постах управления технологическими процессами, в залах вычислительной техники и др.).

Перечень других рабочих мест и видов работ, при которых должны обеспечиваться оптимальные величины микроклимата, определяются Санитарными правилами по отдельным отраслям промышленности и другими документами, согласованными с органами Госсанэпиднадзора.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						50
Изм.	Лист	№ докум.	Подпись	Дата		

Холодный период года – это период года, характеризующийся среднесуточной температурой наружного воздуха, равной $+10^{\circ}\text{C}$ и ниже.

Среднесуточная температура наружного воздуха – средняя величина температуры наружного воздуха, измеренная в определенные часы суток через одинаковые интервалы времени. Она принимается по данным метеорологической службы.

Теплый период года – это период года, характеризующийся среднесуточной температурой наружного воздуха выше $+10^{\circ}\text{C}$.

Допустимые микроклиматические условия установлены по критериям допустимого и функционального состояния человека на период восьмичасовой рабочей смены. Они не столь комфортны, как оптимальные, однако не вызывают повреждений или каких-либо иных нарушений состояний здоровья человека. Однако в ряде случаев такие условия могут привести к возникновению общих или локальных ощущений теплового дискомфорта, напряжению механизмов терморегуляции, ухудшению самочувствия и понижению работоспособности человека.

Допустимые величины показателей микроклимата устанавливаются в случаях, когда по технологическим требованиям, техническим и экономическим причинам не могут быть обеспечены оптимальные величины. В отдельных помещениях депо по ремонту подвижного состава железнодорожного транспорта (например, где производится сушка вагонов) температура воздуха и его влажность не могут быть установлены на уровне оптимальных величин. В противном случае пострадает как сам технологический процесс, так и качество выпущенной продукции.

На практике зачастую бывает так, что в производственных помещениях (опять же из-за технологических требований к производственному процессу) невозможно установить не только оптимальные, но и допустимые нормативные величины показателей микроклимата. В этом случае условия микроклимата следует рассматривать как вредные и опасные. В пример можно привести работы, проводимые, допустим, в лакокрасочных или

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

сталеплавильных цехах различных производственных предприятий. В этом случае для профилактики неблагоприятного воздействия микроклимата на организм работника работодатель должен предпринять определенные меры.

Из выше сказанного следует вывод, что микроклимат на рабочем месте является одним из важнейших факторов, влияющих на работоспособность, как самих ПЭВМ, так и персонала работающего на них. ПЭВМ является очень сложным и дорогостоящим автоматом, имеющим свои рабочие температурные пределы и правила эксплуатации. Для оптимального расчета микроклимата нельзя не учитывать данный фактор, ввиду нормального сосуществования пользователя с самим ПЭВМ.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						52
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения дипломного проекта была изучена предметная область реализации клиентского приложения для обработки потоковых данных, в результате чего было сформулировано техническое задание на разработку программного обеспечения.

Был произведен выбор инструментария, выполнено проектирование, в результате которого были разработаны UML диаграммы и реализовано ПО.

В основной главе дипломного проекта произведено проектирование и реализация клиентской части системы, разработаны макеты страниц приложения, структура базы данных и логика приложения.

Приложение реализовано в соответствии с техническим заданием.

В разделе безопасности жизнедеятельности описан микроклимат помещения для работы с ПЭВМ.

В экономическом разделе был произведен расчет трудоемкости и затрат на разработку программного обеспечения.

Задание на дипломное проектирование выполнено в полном объеме.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Протокол WebRTC на JavaScript [Электронный ресурс] / Р.В. Аристов-электрон. дан. – URL: <http://www.webconnect.edu.ru/vconf/index.php?a=r=thesisDesc> (дата обращения: 15.04.2023).
2. Реализация подсистемы Kurento для корпоративного пользования [Электронный ресурс] / Project Experience. – URL: http://www.prj-exp.ru/structure_of_etl_process.php (дата обращения: 17.04.2023).
3. Барсенин, А. А. Программирование NodeJS. – СПб.: БХВ-Петербург, 2017. – 384с.
4. Полуяров З.В. Использование MySQL для построения хранилищ данных. – Интуит. – 2020. — 187с.
5. Бодрин Д.Н. Корпоративные информационные системы. — Воронеж: Издательство ВГУ, 2021. – с. 231.
6. Лекантов Д.К. Обработка медиа данных в WEB // Системы управления базами данных, 2015. – №3. – с. 40-55.

Приложение А

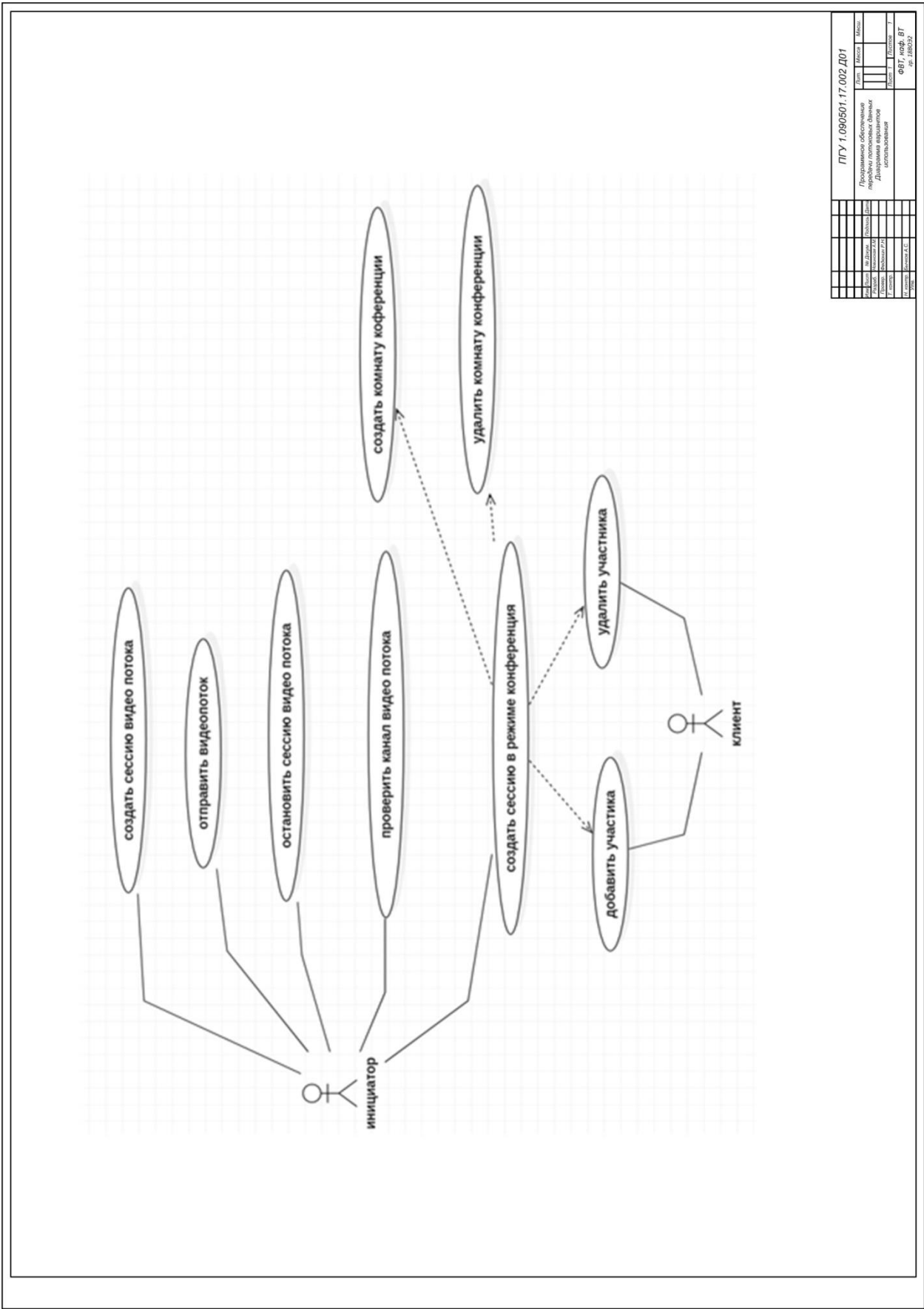
(обязательное)

Программное обеспечение передачи потоковых данных

UML-диаграммы

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						55
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 – Диаграмма вариантов



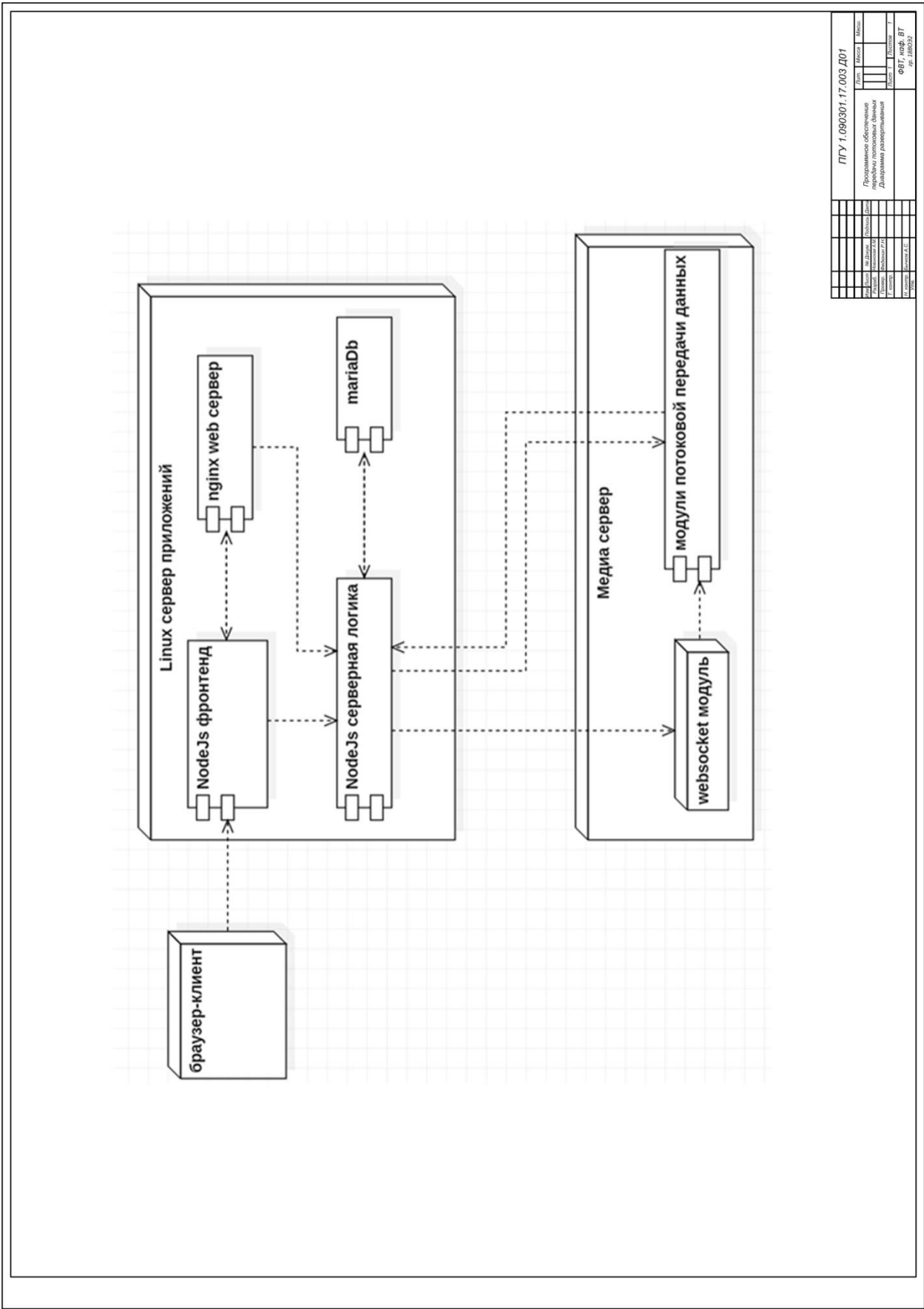
ПГУ 1.090501.17.002 ДО1									
Программа обеспечения программистом Диаграммы вариантов использования									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
ФЭТ, лф. ВТ									
от 18/03/22									

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.09.05.01.08.001 ПЗ

Лист
56

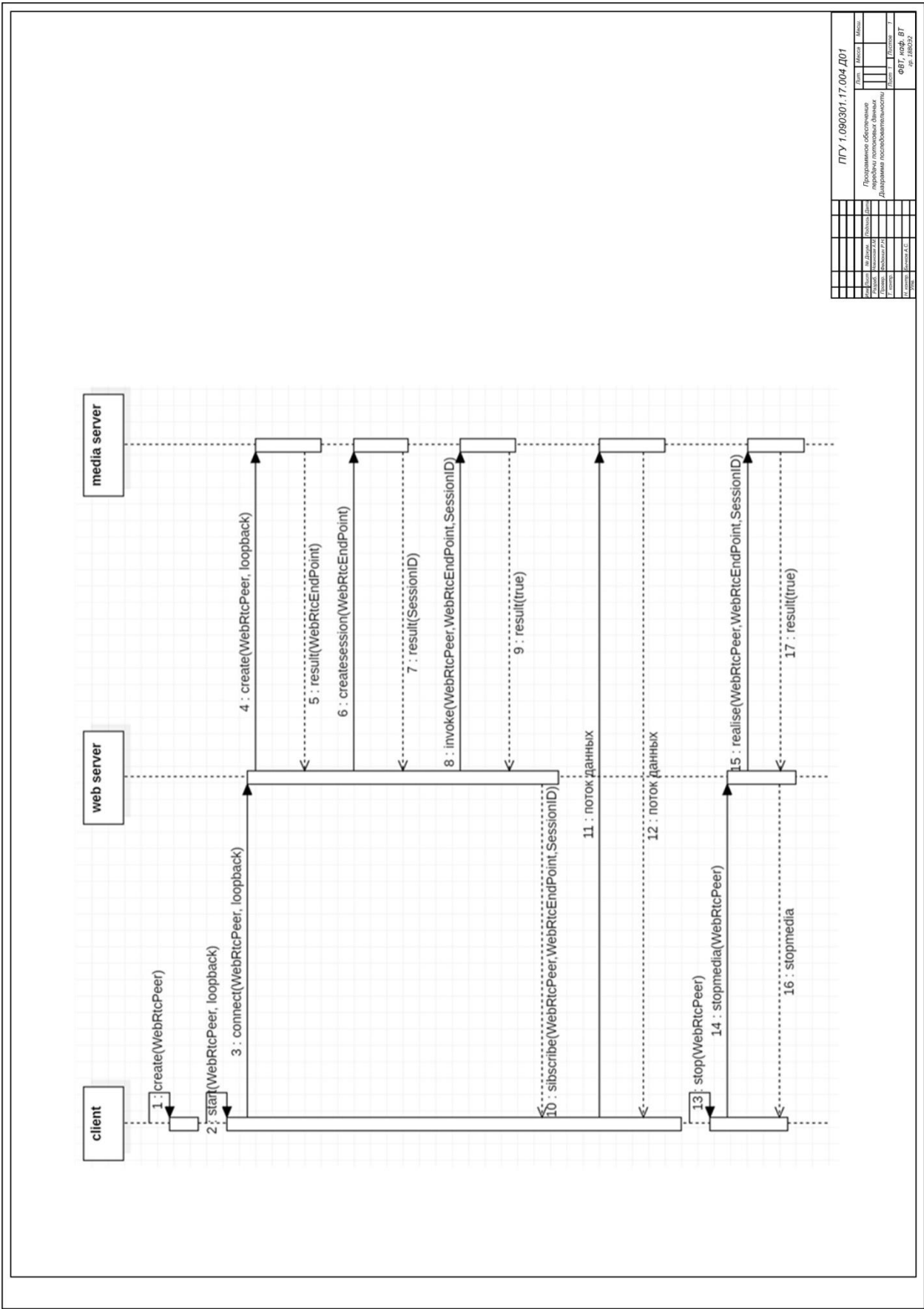
Приложение А.2 – Диаграмма развертывания



ПГУ 1.09.05.01.17.003 ДО1									
Программа обеспечения передачи потоковых данных									
Диаграмма развертывания									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
ФЭТ, подф. ВТ									
от 18/03/22									

Изм.	Лист	№ докум.	Подпись	Дата

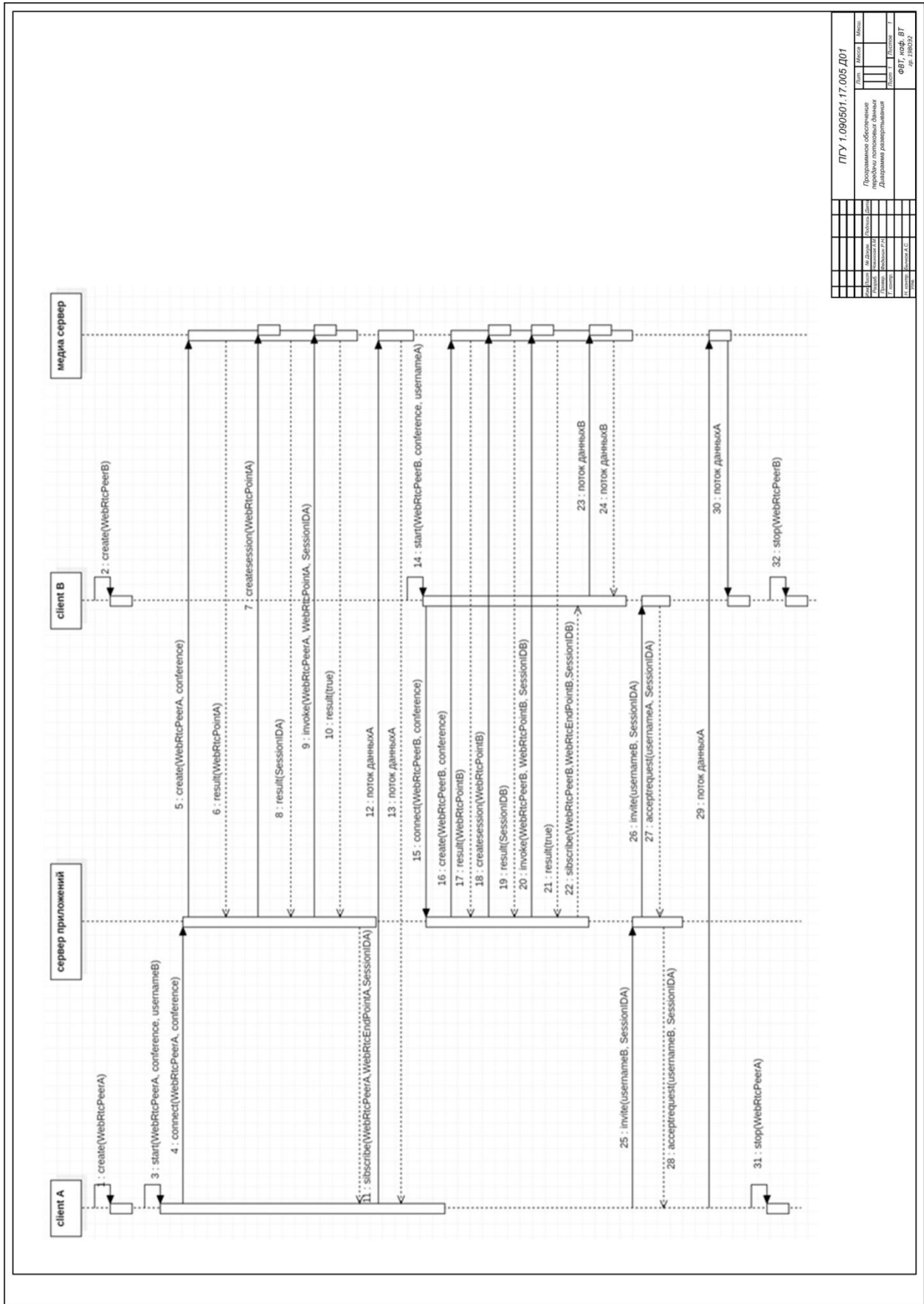
Приложение А.3 – Диаграмма деятельности



ПГУ 1.09.03.01.17.004 ДО1									
Программа обеспечения передачи потоковых данных									
Диаграмма последовательности									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
ФЭТ, код, ВТ									
от 18.03.22									

Изм.	Лист	№ докум.	Подпись	Дата

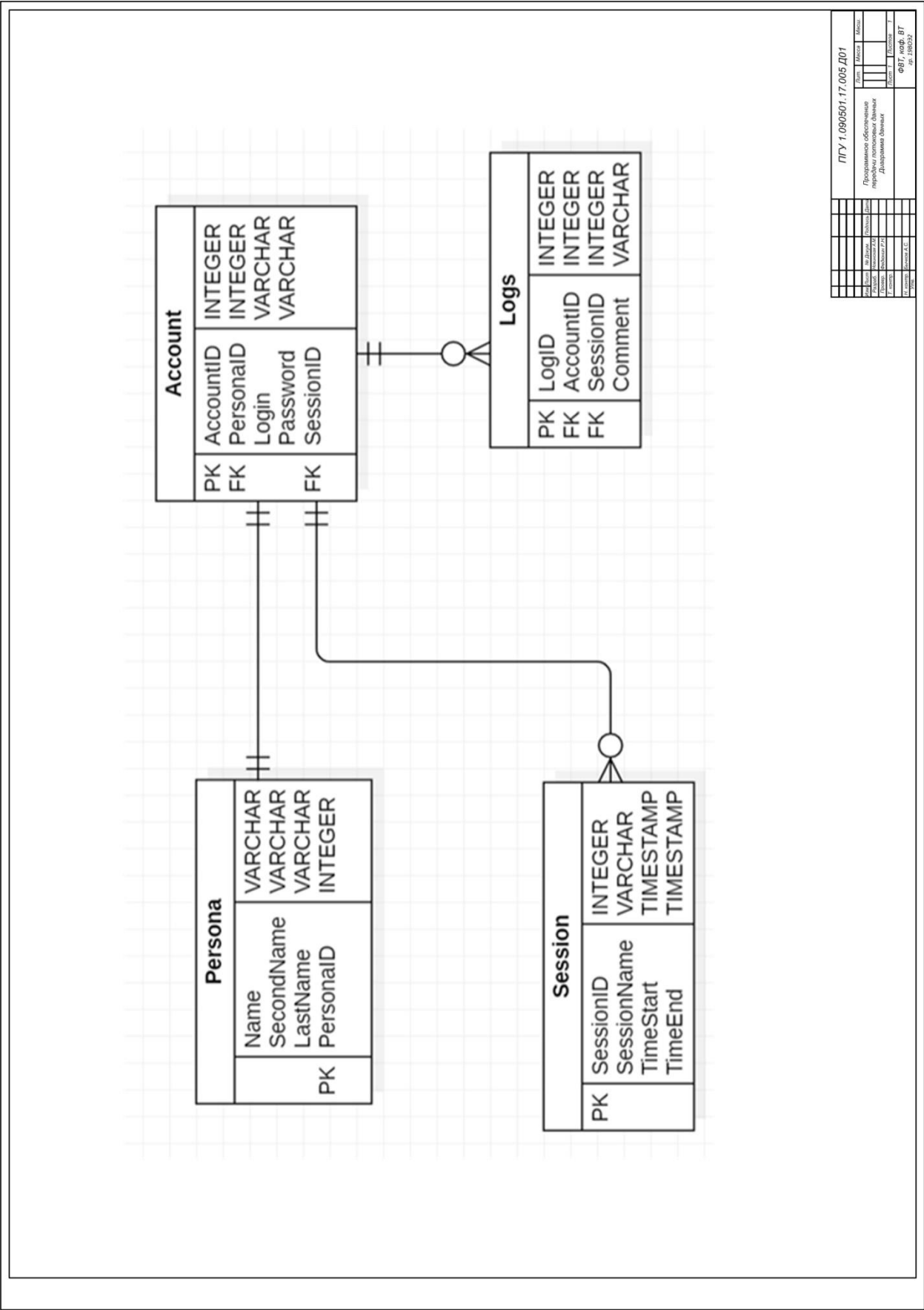
Приложение А.4 – Диаграмма деятельности



ПГУ 1.09.05.01.17.005 ДО1									
Программа обеспечения передачи потоковых данных									
Диаграмма деятельности									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
ФЭТ, под. ВТ									
от 18/03/22									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение А.5 – Диаграмма данных



ПГУ 1.090501.17.005 Д01									
Программное обеспечение прибора полевых данных Датированная версия									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
ФЭТ, код, ВТ									
от 1.09.03									

Изм.	Лист	№ докум.	Подпись	Дата

Приложение Б

(обязательное)

Программное обеспечение передачи потоковых данных

Листинг программы

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Модуль index.js

```
if (typeof kurentoClient == 'undefined')  
window.kurentoClient = require('.');
```

Модуль KurentoClient.js

```
var EventEmitter = require('events').EventEmitter;  
var url = require('url');
```

```
var async = require('async');  
var extend = require('extend');  
var inherits = require('inherits');  
var reconnect = require('reconnect-ws');
```

```
var checkType = require('./checkType');
```

```
var RpcBuilder = require('kurento-jsonrpc');  
var JsonRPC = RpcBuilder.packers.JsonRPC;
```

```
var promiseCallback = require('promisecallback');
```

```
var disguise = require('./disguise')  
var createPromise = require('./createPromise');  
var MediaObjectCreator =  
require('./MediaObjectCreator');  
var TransactionsManager =  
require('./TransactionsManager');
```

```
var TransactionNotCommittedException =  
TransactionsManager  
.TransactionNotCommittedException;  
var transactionOperation =  
TransactionsManager.transactionOperation;
```

```
var MediaObject = require('kurento-client-  
core').abstracts.MediaObject;
```

```
const MEDIA_OBJECT_TYPE_NOT_FOUND = 40100  
const MEDIA_OBJECT_NOT_FOUND = 40101  
const MEDIA_OBJECT_METHOD_NOT_FOUND = 40105  
const INVALID_SESSION = 40007
```

```
const BASE_TIMEOUT = 20000;
```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						62
Изм.	Лист	№ докум.	Подпись	Дата		

```

var PING_INTERVAL = 5000;
var HEARTBEAT = 60000;
var pingNextNum = 0;
var enabledPings = true;
var pingPongStarted = false;
var pingInterval;
var notReconnectIfNumLessThan = -1;

function findIndex(list, predicate) {
    for (var i = 0, item; item = list[i]; i++)
        if (predicate(item)) return i;

    return -1;
};

function serializeParams(params) {
    for (var key in params) {
        var param = params[key];
        if (param instanceof MediaObject || (param &&
        (params.object !==
            undefined ||
        params.hub !== undefined || params.sink !== undefined)))
        {
            if (param && param.id !== null) {
                params[key] = param.id;
            }
        }
    };

    return params;
};

function serializeOperation(operation, index) {
    var params = operation.params;

    switch (operation.method) {
        case 'create':
            params.constructorParams =
                serializeParams(params.constructorParams);
            break;

        default:
            params = serializeParams(params);
            params.operationParams =
                serializeParams(params.operationParams);
    }
}

```

```

};

operation.jsonrpc = "2.0";

operation.id = index;
};

function deferred(mediaObject, params, prevRpc,
callback) {
    var promises = [];

    if (mediaObject != undefined)
promises.push(mediaObject);

    for (var key in params) {
        var param = params[key];
        if (param !== undefined)
promises.push(param);
    };

    if (prevRpc != undefined)
promises.push(prevRpc);

    return promiseCallback(Promise.all(promises),
callback);
};

function noop(error, result) {
    if (error) console.trace(error);

    return result;
};

[проверка связи - поток идет обратно клиенту через
медиа конвертеры]
*/
function KurentoClient(ws_uri, options, callback) {
    if (!(this instanceof KurentoClient))
        return new KurentoClient(ws_uri, options, callback);

    var self = this;

    EventEmitter.call(this);

    //ВЫЗОВ

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						64
Изм.	Лист	№ докум.	Подпись	Дата		

```

varprevRpc = Promise.resolve(); //запрос пошел в сервер
Куренто
var prevRpc_result = Promise.resolve(); //
ответотсервераКуренто

    if (options instanceof Function) {
        callback = options;
        options = undefined;
    };

options = options || {};

var failAfter = options.failAfter
if (failAfter == undefined) failAfter = Infinity

    if (options.enableTransactions === undefined)
options.enableTransactions =
    true
    if (options.strict === undefined) options.strict =
true

options.request_timeout = options.request_timeout ||
BASE_TIMEOUT;
options.response_timeout = options.response_timeout ||
BASE_TIMEOUT;
options.duplicates_timeout = options.duplicates_timeout
|| BASE_TIMEOUT;

var objects = {};

function onNotification(message) {
    var method = message.method;
    var params = message.params.value;

    var id = params.object;

    var object = objects[id];
    if (!object)
        return console.warn("Unknown object id '" + id +
"", message);

    switch (method) {
        case 'onEvent':
object.emit(params.type, params.data);
        break;

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

```

        //      case 'onError':
        default:
console.warn("method '");
    };
};

if (typeof ws_uri == 'string') {
    var access_token = options.access_token;
    if (access_token != undefined) {
        ws_uri = url.parse(ws_uri, true);
        ws_uri.query.access_token = access_token;
        ws_uri = url.format(ws_uri);

        delete options.access_token;
    };
}

var rpc = new RpcBuilder(JsonRPC, options, function
(request) {
    if (request instanceof RpcBuilder.RpcNotification) {
        if (request.duplicated != undefined)

            return onNotification(request);
    };

console.error('Ошибка запроса', request);
    });

    var encodeTransaction = options.enableTransactions ?
commitTransactional :
    commitSerial;

    var transactionsManager = new
TransactionsManager(this,
    function (operations, callback) {
        var params = {
            object: self,
            operations: operations
        };

encodeTransaction(params, callback)
    });

```

```

this.beginTransaction =
transactionsManager.beginTransaction.bind(
    transactionsManager);
this.endTransaction =
transactionsManager.endTransaction.bind(
    transactionsManager);
this.transaction =
transactionsManager.transaction.bind(transactionsManager
);

    Object.defineProperty(this, 'sessionId', {
        configurable: true
    })
this.on('disconnect', function () {
onDisconnected();
    Object.defineProperty(this, 'sessionId', {
        configurable: false,
        get: function () {
            throw new SyntaxError('Клиентнедоступен')
        }
    })

    for (var id in objects)
        objects[id].emit('release')
})

    function onReconnected(sameSession) {
self.emit('reconnected', sameSession);
    }

    function onDisconnected() {
self.emit('disconnected');
    }

    function onConnectionFailed() {
self.emit('connectionFailed');
    }

    function onConnected() {
self.emit('connected');
    }

    function send(request) {
        var method = request.method

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

```

var params = request.params
var callback = request.callback
var stack = request.stack

var requestTimestamp = Date.now()

rpc.encode(method, params, function (error, result) {
  if (error) {
    var responseTimestamp = Date.now()

    var constructor = Error
    switch (error.code) {
      case MEDIA_OBJECT_TYPE_NOT_FOUND:
        constructor = TypeError
        break

      case MEDIA_OBJECT_NOT_FOUND:
        constructor = ReferenceError
        break

      case MEDIA_OBJECT_METHOD_NOT_FOUND:
        constructor = SyntaxError
        break
    }

    error = extend(new constructor(error.message ||
error), error);

    Object.defineProperties(error, {
      'requestTimestamp': {
        value: requestTimestamp
      },
      'responseTimestamp': {
        value: responseTimestamp
      },
      'stack': {
        value: [error.toString()].concat(
error.stack.split('\n')[1],
error.stack.split('\n').slice(2)
).join('\n')
      }
    })
  } else if ((self.sessionId !== result.sessionId)
&& (result.value !==
'pong'))

```

```

        Object.defineProperty(self, 'sessionId', {
            configurable: true,
            value: result.sessionId
        })

callback(error, result);
    });
}

function operationResponse(operation, index) {
    var callback = operation.callback || noop;

    var operation_response = this.value[index];
    if (operation_response == undefined)
        return callback(new Error(
Команданасерверенеотработана'));

    var error = operation_response.error;
    var result = operation_response.result;

    var id;
    if (result) id = result.value;

    switch (operation.method) {
    case 'create':
        var mediaObject = operation.params.object;

        if (error) {
            mediaObject.emit('_id', error);
            return callback(error)
        }

callback(null, registerObject(mediaObject, id));
        break;

        default:
callback(error, result);
    }
}

function sendImplicitTransaction(operations) {
    function callback(error, result) {
        if (error) return console.error('Implicit
transaction failed')

```

```

operations.forEach(operationResponse, result)
    }

operations.forEach(serializeOperation)

    var request = {
        method: 'transaction',
        params: {
            operations: operations
        },
        callback: callback
    }
    send(request)
}

var queueEncode = []

function sendQueueEncode() {
    var request = queueEncode.shift()

    var operations = []

    while (request) {
        if (request.method === 'transaction') {
            if (operations.length) {
                sendImplicitTransaction(operations)
                operations = []
            }

            send(request)
        } else
operations.push(request)

        request = queueEncode.shift()
    }

    if (operations.length)
sendImplicitTransaction(operations)
}

else
    send(request)
}

function encode(method, params, callback) {

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

        var stack = (new Error).stack

params.sessionId = self.sessionId

self.then(function () {
    if (options.useImplicitTransactions&&
!queueEncode.length)
async.setImmediate(sendQueueEncode)

    var request = {
        method: method,
        params: params,
        callback: callback
    }
    Object.defineProperty(request, 'stack', {
        value: stack
    })

    if (options.useImplicitTransactions)
        queueEncode.push(request)
    else
        send(request)
},
callback)
}

function encodeCreate(transaction, params, callback) {
    if (transaction)
        return transactionOperation.call(transaction,
'create', params, callback)

    if (transactionsManager.length)
        return
transactionOperation.call(transactionsManager, 'create',
params, callback);

    callback = callback || noop;

    function callback2(error, result) {
        var mediaObject = params.object;

        if (mediaObject === result) return callback(null,
mediaObject);

        if (error) {

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						71
Изм.	Лист	№ докум.	Подпись	Дата		

```

        mediaObject.emit('_id', error);
        return callback(error);
    }

    var id = result.value;

    callback(null, registerObject(mediaObject, id));
}

return deferred(null, params.constructorParams,
null, function (error) {
    if (error) throw error;

params.constructorParams =
serializeParams(params.constructorParams);

    return encode('create', params, callback2);
})
.catch(callback)
};

var object = params.object;
if (object && object.transactions&&
object.transactions.length) {
    var error = new TransactionNotCommittedException();
    error.method = method;
    error.params = params;

    return setTimeout(callback, 0, error)
};

for (var key in params.operationParams) {
    var object = params.operationParams[key];

    if (object && object.transactions&&
object.transactions.length) {
        var error = new
TransactionNotCommittedException();
        error.method = method;
        error.params = params;

        return setTimeout(callback, 0, error)
    };
}

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

        if (transactionsManager.length)
            return
transactionOperation.call(transactionsManager, method,
    params,
        callback);

    var promise = new Promise(function (resolve, reject)
    {
        function callback2(error, result) {
            if (error) return reject(error);

            resolve(result);
        };

        prevRpc = deferred(params.object,
            params.operationParams, prevRpc,
                function (error) {
                    if (error) throw error

                    params = serializeParams(params);
            params.operationParams = serializeParams(params
                .operationParams);

                    return encode(method, params, callback2);
                })
    }.catch(reject)
    });

    prevRpc_result = promiseCallback(promise, callback);

    if (method == 'release') prevRpc = prevRpc_result;
}

function commitTransactional(params, callback) {
    if (transactionsManager.length)
        return
transactionOperation.call(transactionsManager,
    'transaction',
        params, callback);

    callback = callback || noop;

    var operations = params.operations;

```

```

var promises = [];

function checkId(operation, param) {
    if (param instanceof MediaObject && param.id ===
undefined) {
        var index = findIndex(operations, function
(element) {
            return operation !== element &&
element.params.object === param;
        });

        // MediaObject dependency is created in this
transaction,
        return param
    }

operations.forEach(function (operation) {
    var params = operation.params;

    switch (operation.method) {
        case 'create':
            var constructorParams =
params.constructorParams;
            for (var key in constructorParams)
                constructorParams[key] = checkId(operation,
constructorParams[
                    key]);
            break;

        default:
            params.object = checkId(operation, params.object);

            var operationParams = params.operationParams;
            for (var key in operationParams)
                operationParams[key] = checkId(operation,
operationParams[key]);
    };
});

function callback2(error, result) {
    if (error) return callback(error);

operations.forEach(operationResponse, result)

callback(null, result);

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```

};

Promise.all(promises).then(function () {
operations.forEach(serializeOperation)

encode('transaction', params, callback2);
    },
    callback);
}

function commitSerial(params, callback) {
    if (transactionsManager.length)
        return
transactionOperation.call(transactionsManager,
'transaction',
    params, callback);

    var operations = params.operations;

    async.each(operations, function (operation) {
        switch (operation.method) {
            case 'create':
encodeCreate(undefined, operation.params,
operation.callback);
                break;

            case 'transaction':
commitSerial(operation.params.operations,
operation.callback);
                break;

            default:
encodeRpc(undefined, operation.method, operation.params,
operation.callback);
        }
    },
    callback)
}

function registerObject(mediaObject, id) {
    var object = objects[id];
    if (object) return object;

    mediaObject.emit('_id', null, id);
}

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						75
Изм.	Лист	№ докум.	Подпись	Дата		

```

objects[id] = mediaObject;

mediaObject.once('release', function () {
    delete objects[id];
    if (id.endsWith('_kurento.MediaPipeline')) {
        Object.keys(objects).forEach(function (objectId)
        {
            // If deleting a pipeline from cache, delete
            all children of pipeline, if any remains, as server
            should have removed them also
            if (objectId.startsWith(id))
objects[objectId].emit(
                'release');
        })
    }
});

var mediaObjectCreator = new MediaObjectCreator(this,
encodeCreate,
    encodeRpc, encodeTransaction,
this.getMediaobjectById.bind(this),
options.strict);

function describe(id, callback) {
    if (id == undefined)
        return callback(new TypeError("'id' can't be null
or undefined"))

    var mediaObject = objects[id];
    if (mediaObject) return callback(null, mediaObject);

    var params = {
        object: id
    };

    function callback2(error, result) {
        if (error) return callback(error);

        var mediaObject =
mediaObjectCreator.createImmediate(result);

        return callback(null, registerObject(mediaObject,
id));
    }
}

```

```

encode('describe', params, callback2);
};

Object.defineProperty(this, '_resetCache', {
  value: function () {
    objects = {}
  }
})

function connect(callback) {
  callback = (callback || noop).bind(this)

  //
  // Ping
  //
  function enablePing() {
    enabledPings = true;
    if (!pingPongStarted) {
      pingPongStarted = true;
      pingInterval = setInterval(sendPing, HEARTBEAT);
sendPing();
    }
  }

  function updateNotReconnectIfLessThan() {
    notReconnectIfNumLessThan = pingNextNum;
console.log("notReconnectIfNumLessThan = " +
notReconnectIfNumLessThan);
  }

  function sendPing() {
    if (enabledPings) {
      var params = null;

      if (pingNextNum == 0 || pingNextNum ==
notReconnectIfNumLessThan) {
        params = {
          interval: PING_INTERVAL
        };
      }

      pingNextNum++;

      var request = {

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		

```

        method: 'ping',
        params: params,
        callback: (function (pingNum) {
            return function (error, result) {
                if (error) {
                    if (pingNum > notReconnectIfNumLessThan)
                {
                    enabledPings = false;
                }
                updateNotReconnectIfLessThan();
                console.log(
                    "Server did not respond to ping
                message " +
                    pingNum + "." );
                clearInterval(pingInterval);
                pingPongStarted = false;
            }
        }
    } (pingNextNum))
    }
    send(request);
    } else {
        console.log("Trying to send ping, but ping is not
        enabled");
    }
}

var closed = false;
var reconnected = false;
var re = reconnect({
    // all options are optional
    // initialDelay: 1e3,
    // maxDelay: 30e3,
    // type: 'fibonacci',          // available:
    fibonacci, exponential
    // randomisationFactor: 0,
    // immediate: false
    failAfter: failAfter
}, function (ws_stream) {
    if (closed)
        ws_stream.writable = false;

    rpc.transport = ws_stream;
    enablePing();
    if (reconnected) {

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

```

var params = {
    sessionId: self.sessionId
};
var request = {
    method: 'connect',
    params: params,
    callback: function (error, response) {
        if (error) {
            if (error.code === INVALID_SESSION) {
console.log("Invalid Session")
                objects = {}
                onReconnected(false);
            }
        } else {
            onReconnected(true);
        }
    }
};
send(request);
} else {
onConnected();
}
})
.connect(ws_uri, options.socket);

Object.defineProperty(this, '_re', {
    get: function () {
        return re
    }
});

this.close = function () {
    closed = true;

    enabledPings = false;
    clearInterval(pingInterval);
    pingPongStarted = false;

    prevRpc_result.then(re.disconnect.bind(re));
};

re.on('fail', this.emit.bind(this, 'disconnect'));

re.on('reconnect', function (n, delay) {

```

```

console.log('reconnect to server', n, delay,
self.sessionId);
    if (pingInterval != undefined) {
        clearInterval(pingInterval);
        pingPongStarted = false;
    }

    reconnected = true;
})

this.then = function (onFulfilled, onRejected) {
    if (re.connected)
        var promise =
Promise.resolve(disguise.unthenable(this))
    else if (!re.reconnect)
        var promise = Promise.reject(new
Error('Connection error'))
    else {
        var self = this

        var promise = new Promise(function (resolve,
reject) {
            function success() {
re.removeListener('fail', failure);

                resolve(disguise.unthenable(self));
            };

            function failure() {
            };

re.once('connection', success);
re.once('fail', failure);
        });

    }

    promise = promise.then(onFulfilled ?
onFulfilled.bind(this) :
        function (result) {
            return Promise.resolve(result)
        },
onRejected ? onRejected.bind(this) :
        function (error) {
            return Promise.reject(error)
        }
    );

```

```

        });

        return disguise(promise, this)
    };

    this.catch = this.then.bind(this, null);

    var thenable = this
    if (options.strict)
        thenable = this.getServerManager()
    .then(function (serverManager) {
        return serverManager.getInfo()
    })
    .then(function (info) {
        var serverModules = info.modules.map(function
(module) {
            return module.name
        })

        var notInstalled =
KurentoClient.register.modules.filter(
            function (module) {
                return serverModules.indexOf(module) < 0
            })

        connect.call(self, callback);
    });
    inherits(KurentoClient, EventEmitter);
    /**
     * @callback
     module:kurentoClient.KurentoClient~constructorCallback
     * @param {external:Error} error
     * @param {module:kurentoClient.KurentoClient} client
     KurentoClient.prototype.connect = function (media,
callback) {
        if (!(media instanceof Array)) {
            media = Array.prototype.slice.call(arguments, 0);
            callback = (typeof media[media.length - 1] ===
'function') ? media.pop() :
                undefined;
        }

        callback = (callback || noop).bind(this)
        if (media.length < 2)

```

```

        throw new SyntaxError("Need at least two media
elements to connect");

    return media[0].connect(media.slice(1), callback)
};

KurentoClient.prototype.getServerManager = function
(callback) {
    return
this.getMediaobjectById('manager_ServerManager',
callback)
};

var singletons = {};

KurentoClient.getSingleton = function (ws_uri, options,
callback) {
    var client = singletons[ws_uri]
    if (!client) {
        if (options instanceof Function) {
            callback = options;
            options = undefined;
        }

        client = KurentoClient(ws_uri, options, function
(error, client) {
            if (error) return callback(error);

            singletons[ws_uri] = client
client.on('disconnect', function () {
            delete singletons[ws_uri]
        })
    });
    }

    return disguise(promiseCallback(client, callback),
client)
}

KurentoClient.getComplexType = function (complexType) {
    return
KurentoClient.register.complexTypes[complexType]
};

```

```

module.exports = KurentoClient;

Модуль MediaObjectCreator.js

var async = require('async');

var checkType = require('./checkType');
var checkParams = checkType.checkParams;
var extend = require('extend');

var createPromise = require('./createPromise');
var register = require('./register');

var Transaction =
require('./TransactionsManager').Transaction;

function getConstructor(type, strict) {
    var result = register.classes[type.qualifiedType] ||
register.abstracts[type
.qualifiedType] ||
register.classes[type.type] ||
register.abstracts[type.type] ||
register.classes[type] || register.abstracts[type];
    if (result) return result;

    if (type.hierarchy != undefined) {
        for (var i = 0; i <= type.hierarchy.length - 1; i++)
        {
            var result = register.classes[type.hierarchy[i]]
|| register.abstracts[
type.hierarchy[i]];
            if (result) return result;
        };
        if (strict) {
            var error = new SyntaxError("Unknown type '" + type
+ "'")
            error.type = type

            throw error
        }
    }

    console.warn("Unknown type '", type, "'", using
MediaObject instead);
    return register.abstracts.MediaObject;
}

```

```

};

function createConstructor(item, strict) {
    var constructor = getConstructor(item, strict);

    if (constructor.create) {
        item = constructor.create(item.params);
        var prototype = constructor.prototype;
        inherits(constructor, getConstructor(item, strict));
        extend(constructor.prototype, prototype);
    };

    constructor.item = item;

    return constructor;
}

var checkMediaElement = checkType.bind(null,
'MediaElement', 'media');

function MediaObjectCreator(host, encodeCreate,
encodeRpc, encodeTransaction,
describe, strict) {
    if (!(this instanceof MediaObjectCreator))
        return new MediaObjectCreator(host, encodeCreate,
encodeRpc,
encodeTransaction, describe)

function createObject(constructor) {
    var mediaObject = new constructor(strict)

    mediaObject.on('_describe', describe);
    mediaObject.on('_rpc', encodeRpc);

    if (mediaObject instanceof
register.abstracts['kurento.Hub'] ||
mediaObject instanceof register
.classes['kurento.MediaPipeline'])
        mediaObject.on('_create', encodeCreate);

    if (mediaObject instanceof
register.classes['kurento.MediaPipeline'])
        mediaObject.on('_transaction', encodeTransaction);

    return mediaObject;
}

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

```

};

function createMediaObject(item, callback) {
    var transaction = item.transaction;
    delete item.transaction;

    var constructor = createConstructor(item, strict);

    item = constructor.item;
    delete constructor.item;

    var params = item.params || {};
    delete item.params;

    if (params.mediaPipeline == undefined && host
instanceof register.classes
.MediaPipeline)
params.mediaPipeline = host;

    var params_ = extend({}, params)
item.constructorParams = checkParams(params_,
constructor.constructorParams,
item.type);

    if (Object.keys(params_)) {
item.properties = params_;
    }

    if (!Object.keys(item.constructorParams).length)
        delete item.constructorParams;

    try {
        var mediaObject = createObject(constructor)
    } catch (error) {
        return callback(error)
    };

    Object.defineProperty(item, 'object', {
        value: mediaObject
    });

    encodeCreate(transaction, item, callback);

    return mediaObject

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

```

this.create = function (type, params, callback) {
    var transaction = (arguments[0] instanceof
Transaction) ? Array.prototype
.shift.apply(arguments) : undefined;

    switch (arguments.length) {
    case 1:
        params = undefined;
    case 2:
        callback = undefined;
    };

    if (params instanceof Function) {
        if (callback)
            throw new SyntaxError("Nothing can be defined
after the callback");

        callback = params;
        params = undefined;
    };

    if (type instanceof Array) {
        var createPipeline = false

type.forEach(function (request) {
    var params = request.params || {}

    if (typeof params.mediaPipeline === 'number')
        createPipeline = true
    })

    function connectElements(error, elements) {
        if (error) return callback(error)

        if (params === true && host.connect)
            return host.connect(elements.filter(function
(element) {
                try {
                    checkMediaElement(element)
                    return true
                } catch (e) {}
            })),
            function (error) {
                if (error) return callback(error)

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						86
Изм.	Лист	№ докум.	Подпись	Дата		

```

callback(null, elements)
    })

callback(null, elements)
    }

    if (createPipeline)
        return host.transaction(function () {
            var mediaObjects = []

async.map(type, function (request, callback) {
            var params = request.params || {}

                if (typeof params.mediaPipeline ===
'number')
params.mediaPipeline = mediaObjects[params
.mediaPipeline]

mediaObjects.push(createMediaObject(request, callback))
            },
            connectElements)
        })

        return createPromise(type, createMediaObject,
connectElements)
    }

    type = {
        params: params,
        transaction: transaction,
        type: type
    };

    return createMediaObject(type, callback)
};

this.createImmediate = function (item) {
    var constructor = createConstructor(item, strict);
    delete constructor.item;

    return createObject(constructor);
}
}

```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		

```
module.exports = MediaObjectCreator;
```

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						88
Изм.	Лист	№ докум.	Подпись	Дата		

ПриложениеВ

(обязательное)

Программное обеспечение передачи потоковых данных

Презентация

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						89
Изм.	Лист	№ докум.	Подпись	Дата		

Программное обеспечение передачи потоковых данных

Автор работы:

студент гр. 19ВОЭ1 Новинская Алина Михайловна

Научный руководитель:

к.т.н., доцент Федюнин Роман Николаевич

Пенза, 2023

Цель работы и объекты

Цель работы – Разработать клиентское приложение для платформы Kurento (платформа работы с потоковыми медиа данными).

Объектом проекта является клиентское приложение, выполняющие следующие функции:

- подключение к платформе обработки потоковых данных Kurento;
- выбор доступных режимов работы;
- работа в режиме тестирования канала связи;
- работа в режиме конференции с несколькими участниками;
- завершение сеанса связи.

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						90
Изм.	Лист	№ докум.	Подпись	Дата		

Задачи приложения

Программа реализована в виде клиентского приложения на базе протокола WebRTC.

Передача и прием данных клиентским приложением осуществляется за счет подключения соответствующих модулей медиа-платформы.

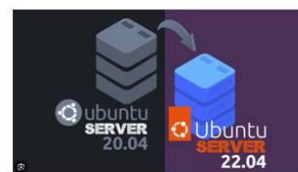
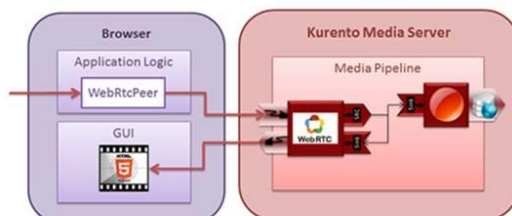
Сравнение аналогов

Возможности	Контур. Толк.	TrueConf	Яндекс. Телемост	Видеозвонки. Mail.ru
Запись разговоров	имеется	имеется		
Работа с разных устройств	да	да		
Поиск	да	нет	да	да
Текстовый чат	нет	нет	да	да
Поддерживаемые Платформы	Windows Linux, Android Mac IOS WEB	Windows Android Mac IOS WEB	Windows, Linux, Android IOS WEB	Android IOS WEB
Развёртывание	облачное	облачное, локальное	облачное	облачное
Открытый код	нет	нет	нет	нет
Бесплатное пользование	нет	нет	да	да

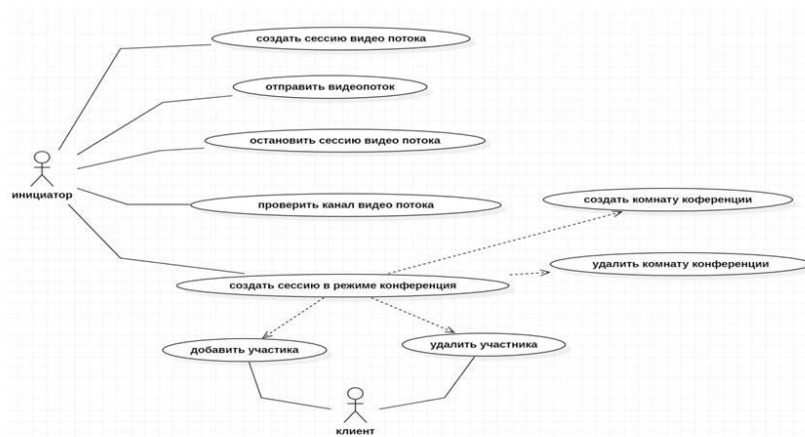
Аналоги обладают недостатками:

облачные решения, которые не позволяют развернуть приложение на аппаратной платформе заказчика и полностью ее контролировать; закрытые системы с отсутствием доступа к исходным кодам; не все реализованное программное обеспечение кроссплатформенное.

Стек технологий



Основные функции приложения



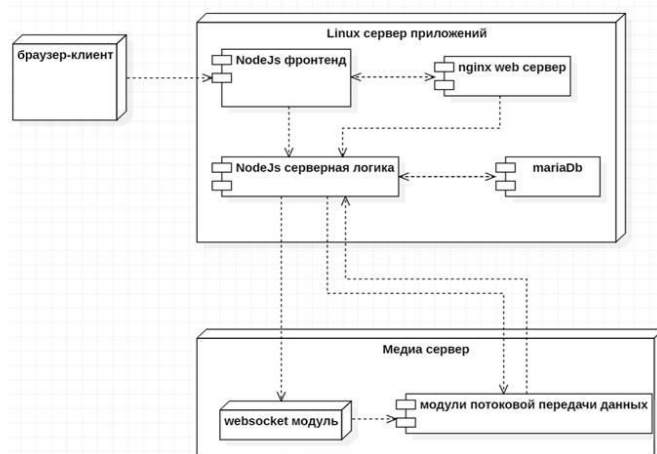
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ1.09.05.01.08.001 ПЗ

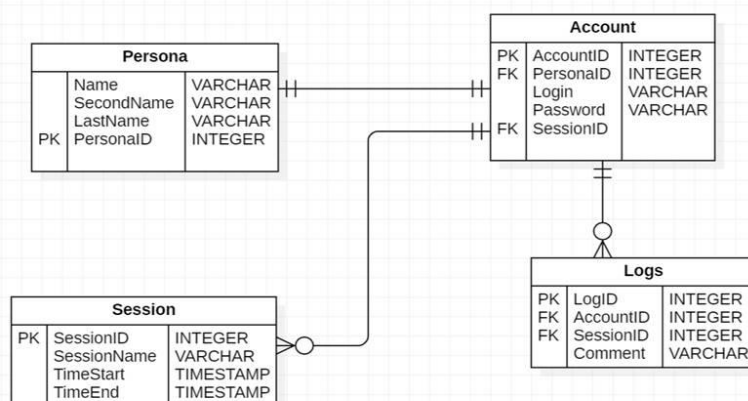
Лист

92

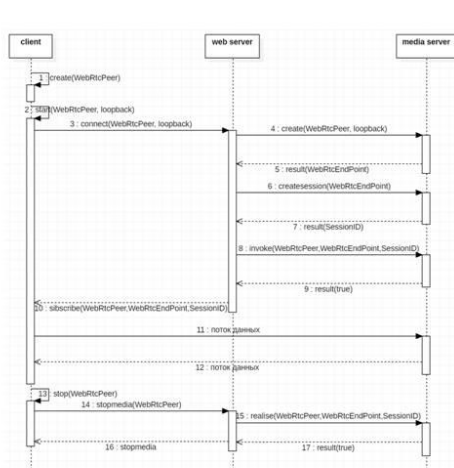
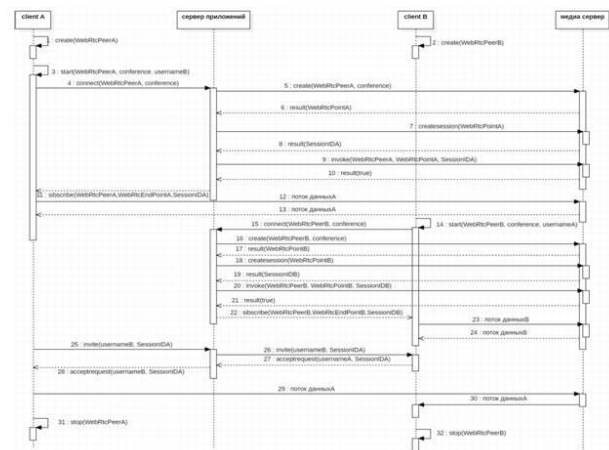
Архитектура приложения



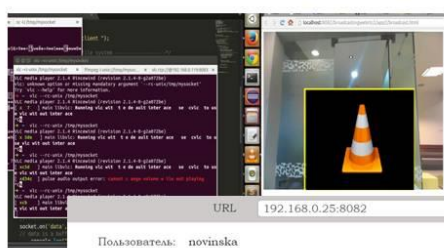
База данных приложения



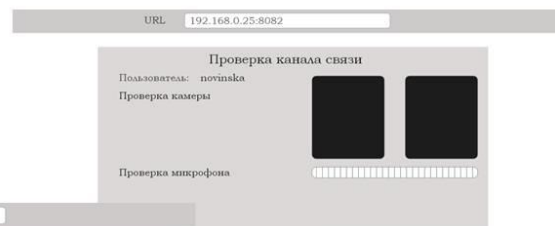
Работа приложения в различных режимах



Скриншоты запущенного приложения



Доступные режимы работы



Программное обеспечение передачи потока данных



Изм.	Лист	№ докум.	Подпись	Дата

Результаты работы

- Разработано приложение полностью соответствующее техническому заданию;
- Проведено функциональное тестирование (успешно);
- Намечены задачи по развитию функциональных возможностей программы.

Спасибо за внимание!

					ПГУ1.09.05.01.08.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		