**Введение**

Целью данной курсовой работы является разработка приложения «Видео чат».

Видео чат – это приложение, обеспечивающее голосовое общение между людьми с поддержкой видео режима. Для передачи видео и аудио потоков часто используется специальные протоколы с шифрованием или без. Данное приложение позволяет возможности регистрации новых пользователей, добавлять новые контакты, а также совершать звонки и участвовать в видео конференциях. На текущий момент существует немного приложений предоставляющих такие возможности. Одно из самых известных приложений «Skype» пользуется своей популярностью благодаря поддержи протокола с шифрованием. «Видео чат» работает через протокол RTP, а так же использует современные технологии обработки видео и аудио.

**Анализ исходных данных**

**1.1 Характеристики требований на разработку**

В данном курсовом проекте необходимо разработать приложение «Видео чат». Для реализации поставленной задачи, нужно создать сервер, который будет принимать запросы от пользователей, и обрабатывать их, а так же клиентское приложение, которое будет предоставлять удобный интерфейс для работы с приложением. Для хранения информации о контактах, пользователях, подключениях нужно разработать специальный модуль, который будет делаться на две части: веб-аппликация и REST – сервисы.

**1.2 Анализ предметной области**

Поскольку необходимо разработать приложение «Видео чат», необходимо выделить все основные задачи и сгруппировать их по типам. Выделим два основных типа задач: клиентские и серверные. В клиентские задачи входит работа с устройствами для захвата видео и аудио, возможность авторизироваться в системе, добавлять новые контакты и совершать звонки. Так же должна быть возможноть для выбора и настройке устройств. Исходя из выше перечисленных задач можно проанализировать существующий пути решения и выбрать наиболее подходящие. Для работы с устройствами выбран «JMF framework». Такой выбор обоснован тем, что JMF полностью написан на java и не использует других библиотек, а так же встроенная поддержка RTP позволяет отсылать видео и аудио потоки по сети. Пользовательский интерфейс будет разрабатываться с использование Swing.

Swing — библиотека для создания графического интерфейса для программ на языке Java. Он содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д. Swing реализует известный шаблон проектирования c основными принципами UI. Этот шаблон проектирования называется Модель-Представление-Контроллер (Model-View-Controller - MVC) и стремится "разделить роли". MVC хранит код, ответственный за внешний вид чего-нибудь, отдельно от кода, обрабатывающего данные, и отдельно от кода, реагирующего на взаимодействие и выполняющего изменения. Для использования данного фреймворка не требуется сторонних библиотек, т.к. swing поставляется вместе с jdk.

Теперь рассмотрим задачи, которые относятся к задачам серверного типа. Для хранения информации о пользователях, контактах и подключениях, нужно хранилище. Для этой задачи будет использоваться Mongodb.

MongoDB — документо-ориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Написана на языке C++ и распространяется в рамках лицензии Creative Commons.

Для предоставления возможности регистрации новых пользователей, нужно разработать веб-аппликацию, которая будет развернута в каком либо контейнере (для данной задачи выбран TomCat).

Для взаимодействия пользователя с сервером была выбрана архитектура REST.

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.иц. Написана на языке C++ и распространяется в рамках лицензии Creative Commons.

Одной из самых сложных и важных задач является разработка архитектуры модуля организации звонков между пользователями. Вся архитектура базируется на уже существующих паттернах взаимодействия. Для данной задачи подходит паттерн «Request Reply» или так же известный «Exchange Pattern». Реализация данного паттерна позволит организовать процесс звонка – ответа. В качестве брокера будет выбран ActiveMQ.

ActiveMQ — это message broker с открытым исходным кодом (распространяется под лицензией Apache 2.0), который полностью реализует Java Message Service 1.1 (JMS). Он обеспечивает «Enterprise Features», такие как кластеризация, хранение сообщений, с возможностью использовать различные базы данных, кэширование и ведение журналов.

Исходя из выше перечисленных задач и методов их решения можно выделить основные модули:

|  |  |
| --- | --- |
| Название модуля | Опсиание |
| phantombox-domain | Содержит все необходимые модели |
| phantombox-server | Веб-аппликация и REST |
| phantombox-broker | Модуль звонков |
| phantombox-client | Клиент |

Таблица 1 – Основные модули приложения

**1.3 Специальное и общесистемное программное обеспечение**

Для разработки проекта использовалась IntelliJ IDEA, как среда разработки и Gradle для сборки проекта. После компиляции и сборки проекта создается исполняемый «jar» файл, для запуска которого нужна установка jdk 1.7. В качестве хранилища используется Mongodb, ActiveMQ –брокер, TomCat – веб-контейнер.

Для сборки модулей и загрузки артефактов в локальный репозиторий нужно выполнить команду «gradle clean build UA», ниже перечислен порядок сборки.

1. phantombox-domain
2. phantombox-server
3. phantombox-broker
4. phantombox-client

После сборки «phantombox-server» будет создано два основных артефакта: «phantombox-server-web» и «phantombox-server-rest-web», которые нужно развернуть в TomCat. Для этого нужно скопировать данные jar файлы в папку webapps TomCat. После этого запустить сервер.

Для запуска MongoDB ниже описаны основные шаги:

1. Создать новый файл в папке bin MongoDB и назовать его mongodb.config
2. Добавить в mongodb.config одну строку: dbpath=ПУТЬ\_КУДА\_НУЖНО\_СОХРАНЯТЬ\_ФАЙЛЫ\_БАЗЫ\_ДАННЫХ. Например, в Windows можно написать dbpath=c:\mongodb\data а в Linux — dbpath=/etc/mongodb/data.
3. Запустить mongod с параметром --config /path/to/your/mongodb.config.

Для запуска ActiveMQ нужно запустить activemq.bat без всяких дополнительных действий.

После нужно собрать клиента и запустить.

# **2. Программное проектирование**

**2.1 Информационные структуры**

Разрабатываемое приложение осуществляет процессы по обработке некой информации. Исходя из предыдущей главы, уже известны базовые процессы и взаимодействия, теперь нужно выяснить какие модели будут участвовать в процессах. Ниже в таблице перечислены основные модели:

|  |  |
| --- | --- |
| Модель | Описание |
| User | Информация о пользователе |
| Contact | Информация о контакте |
| Connection | Информация о соединении |
| Device | Информация об устройствах (беб-камеры, микрофоны) |
| Setup | Настройки программы |

Таблица 2 – Информационные структуры

Это основные модели, которые участвуют в взаимодействии разных модулей.

## 2.2 Математическое обеспечение

Под математическим обеспечением часто понимают аппаратное обеспечение. Для работы с устройствами используется JMF, его можно загрузить с официального сайта. После чего нужно установить продукт и запустить «jmfinit.exe», JMF найдет все устройства в системе и сконфигурирует файл «jmf.properties» который будет находится в папке lib. Нужно скопировать этот файл в папку «phantombox-client\phantombox-client-ui\src\main\resources\», после чего можно собрать и запустить приложение.

Наличие установленного «JMF» в системе обязательно.

Для сборки используется Gradle build tool. Хорошая и удобная утилита для сборки проекта. TomCat, ActiveMQ и MongoDB так же должны быть установлены в системе. Все эти приложения можно найти на официальных ресурсах.

## 2.3 Функциональное обеспечение

Каждый модуль приложения предоставляет некую функциональность. В таблице ниже перечислены основные модули и их функции.

|  |  |
| --- | --- |
| Модуль | Функции |
| phantombox-server | Сохранение и получение пользователей. Создание нового пользователя и авторизация, получение ip адреса, добавление новых контактов. |
| phantombox-broker | Отправка запроса на видео звонок и получение ответа |
| phantombox-client | Авторизация, , добавление и удаление контактов, настройка устройств, звонки, информация о приложении. |

Таблица 3 – Функциональное обеспечение

Функциональные возможности модулей зависят от типов задач, которые они решают.

## 2.4 Алгоритмическое обеспечение

В данной главе будет рассмотрен алгоритм захвата потока с устройств и отправка его по протоколу RTP. Как уже упоминалось JMF поддерживает протокол RTP, что позволяет пользоваться уже готовой реализацией протокола RTP.

Протокол RTP (англ. Real-time Transport Protocol) работает на транспортном уровне и используется при передаче трафика реального времени.

Протокол RTP переносит в своём заголовке данные, необходимые для восстановления голоса или видеоизображения в приёмном узле, а также данные о типе кодирования информации (JPEG, MPEG и т. п.). В заголовке данного протокола, в частности, передаются временная метка и номер пакета. Эти параметры позволяют при минимальных задержках определить порядок и момент декодирования каждого пакета, а также интерполировать потерянные пакеты.

Спецификация RTP описывает два под-протокола:

* Протокол передачи данных, RTP, который взаимодействует с передачей данных реального времени. Информация, предоставляемая посредством этого протокола включает тайм-стемп (для синхронизации), последовательный номер (для детектирования потери и дублирования пакетов) и формат полезной нагрузки, который определяет формат кодирования данных.
* Протокол контроля, RTCP, используемый для определения качества обслуживания ([QOS](http://ru.wikipedia.org/wiki/QOS)), обратной связи и синхронизации между медиа-потоками. Занимаемая полоса пропускания RTCP — мала в сравнении с RTP, обычно около 5 %.
* Управляющий сигнальный протокол, такой как [SIP](http://ru.wikipedia.org/wiki/SIP), [H.323](http://ru.wikipedia.org/wiki/H.323), [MGCP](http://ru.wikipedia.org/wiki/MGCP) или [H.248](http://ru.wikipedia.org/wiki/H.248). Сигнальные протоколы управляют открытием, модификацией и закрытием RTP-сессий между устройствами и приложениями реального времени.
* Управляющий протокол описания медиа, такой как [Session Description Protocol](http://ru.wikipedia.org/wiki/Session_Description_Protocol" \o "Session Description Protocol).

На рисунке 1 изображена архитектура RTP протокола.

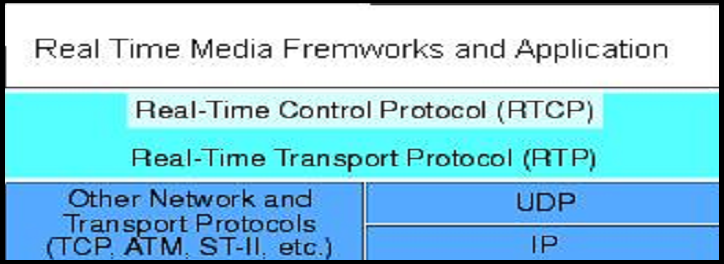


Рис. 1 – Архитектура RTP протокола

JMF 1.0 API (Java Media Player API) позволили программистам разрабатывать Java программы, которые работают с медиа-данными. JMF 2.0 API расширяет возможности получения и хранения медиа-данных, управления типом обработки, которая выполняется во время воспроизведения, а также выполнение кастомизированных обработок над медиа-потоками данных. Кроме того, JMF 2.0 определяет плагин API, который позволяет опытным разработчикам и поставщикам технологий с легкостью настраивать и расширять функциональность JMF.

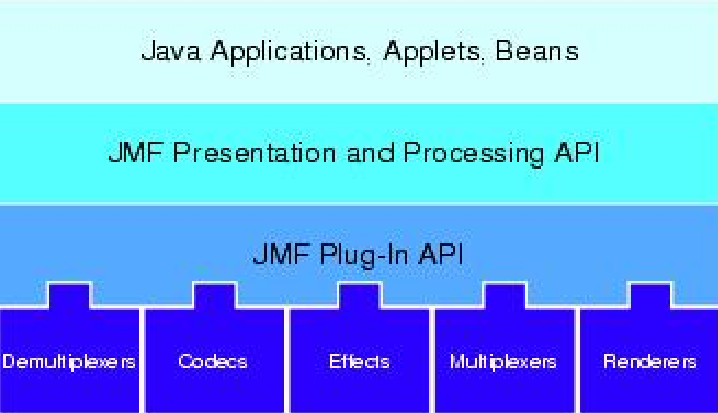


Рис. 2 – Архитектура JMF

Изучив документацию и API «JMF» можно выделить основные компоненты:

DataSource – источник данных. Может быть создан из MediaLocator, который в свою очередь является метаинформацией об устройстве, и создается на основе CaptureDeviceInfo.

Processor – обработчик медиа-данных. Этот компонет выполняет самую основную работу. Он получает данные из DataSource и обрабатывает их в зависимости от типа данных.

Transmitter – передачик. Существует два типа передатчиков: DataSink и RTPM.

Ниже на рисунке изображена общая схема взаимодействия компонентов.

Исходя из выше перечисленного можно выделить следующие шаги к конфигурации компонентов.

1. Получить все устройсва в сети. Для этого нужно воспользоваться CaptureDeviceManager.
2. Выбрать нужное устройство и создать MediaLocator
3. На основе MediaLocator создать DataSource
4. Передать DataSource в Processor и обработать медиа данные.
5. Получить DataOutput из Processor и начать передечу данных.

Ниже изображена схема отправителя медия-контента.

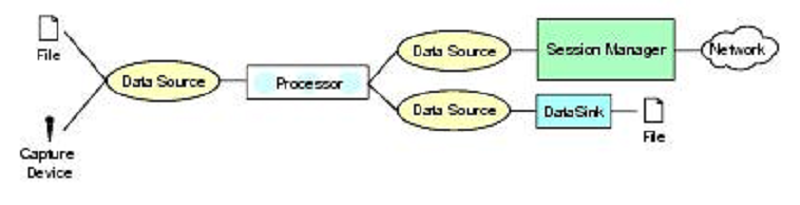


Рис. 3 – Отправитель

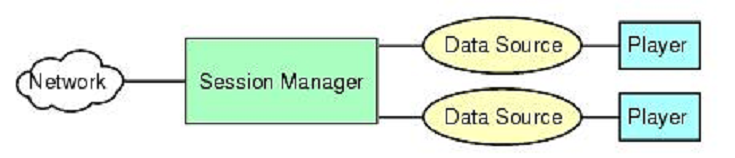


Рис. 4 – Получатель

JMF предоставляет удобный API, который позволяет управлять процессами максимально эффективно. В приложении были реализованы специальные менеджеры, которые отвечают за настройку и конфигурацию компонентов.

## 2.5 Пользовательский интерфейс

Пользовательский интерфейс должен отвечать следующим требованиям:

1. Простота.
2. Удобство.
3. Прямолинейность.
4. Надежность.

По функциональным требованиям интерфейс должен отвечать следующим критериям:

1. Авторизация.
2. Добавление и удаление новых контактов.
3. Выбор нужных устройств (веб-камера и микрофон).
4. Функции звонка и ответа.
5. Выход и приложения.
6. Информация.

# **3 Программная реализация**

## 3.1 Детальная реализация функциональных частей программного обеспечения

В данной главе будут рассмотрены основные аспекты реализации функциональных частей. Так же будут описаны способы взаимодействия с функциональными частям приложения.

#### **3.1.1 Реализация программной части**

Приложение состоит из трех основных модулей: client, sever, broker, каждый из которых так же разбит на подмодули. Ниже на диаграмме изображены основные компоненты сервера.

**SERVER**

**Storage**

**server-rest-call**

**server-storage**

**server-service**

**TomCat**

web application server

**server-rest-web**

**server-web**

Клиент отправляет REST запросы на сервер

**CLIENT**

Рис. 5 – Диаграмма сервера

**Сервер предоставляет REST API для взаимодействия с ним. А так же веб-аппликацию для регистрации новых пользователей. В таблице ниже перечислены возможные REST запросы.**

|  |  |  |
| --- | --- | --- |
| Запрос | Параметры запроса | Описание |
| /user /login | @RequestParam String name, @RequestParam String password, @RequestParam(required = false) String ipAddress | Авторизует пользователя |
| /user /logout | @RequestParam String name, @RequestParam String password | Выход из системы |
| /user/addContactByName | @RequestParam String name, @RequestParam String password, @RequestParam String contactName | Добавляет новый контакт по имени в список контактов |
| /user /addContactByEmail | @RequestParam String name, @RequestParam String password, @RequestParam String email | Добавляет новый контакт по email в список контактов |
| /user /removeContact | @RequestParam String name, @RequestParam String password, @RequestParam String contactId | Удаляет контакт из списка контактов |

**Модуль broker реализует паттерн** «Request Reply» для предоставления функции звонка. Ниже на рисунке изображена общая диаграмма паттерна и схематичное изображение процесса звонка.

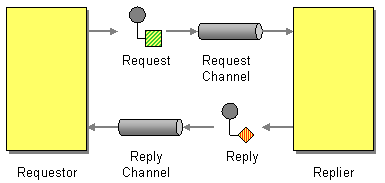


Рис. 6 – Паттерн «Request Reply»

Request queue

Request queue

REQUEST

RESPONSE

Рис. 7 – Схематичное отображение процесса звонка

Схема выше, наглядно показывает процесс звонка. Можно выделить следующие основные этапы.

1. Пользователь отправляет сообщение на брокер с параметром «contactId» собеседника. Каждый пользователь слушает очередь с селектором на свой уникальный id в MongoDB. После отправки сообщение кладется в пул запросов.
2. Получатель получает запрос, составляет ответ и отсылает его с нужным идентификатором. В ответе указывается тип ответа: CANCEL, ACCEPT, REJECT.
3. Инициатор звонка получает ответ и анализирует и удаляет запрос из пула.

Данная задача была выполнена при использовании ActiveMQ и JMS клиента от Spring.

#### **3.1.2 Реализация пользовательского интерфейса**

После запуска клиентского приложения вызывается главное окно программы, которое изображено на рисунке 8.

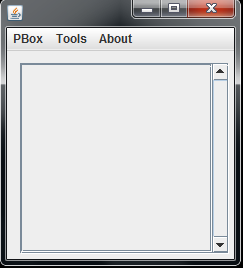


Рис. 8 – Главная форма программы

После чего пользователь должен авторизироваться. Для этого предназначена форма «Login» изображенная на рисунке ниже.

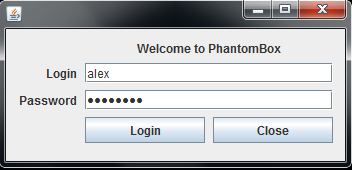


Рис. 9 – Форма авторизации

После авторизации будут загружены контакты пользователя и выдан ip и порты для видео и аудио потоков.

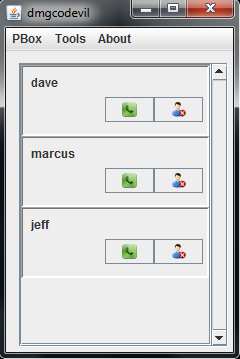


Рис. 10 – Главная форма после авторизации

Теперь пользователь может совершать звонки. Для этого нужно нажать на значок в виде трубки рядом с контактом. После нажатия на кнопку звонка появится следующее окно.



Рис. 11 – Звонок Джефу

В свою очередь собеседник увидит окно с запросом на разговор.

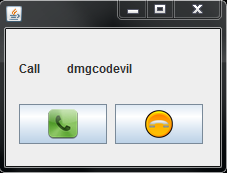


Рис. 12 – Запрос на видео чат

Если собеседник отказывается от разговора, то звонящий абонент увидит форму с сообщением изображенную ниже.

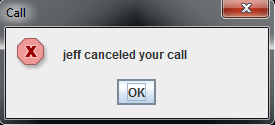


Рис. 13 – Отказ в видео чате

Если собеседник подтверждает запрос, то появляется форма с видео, которое передается от абонента, который звонит.

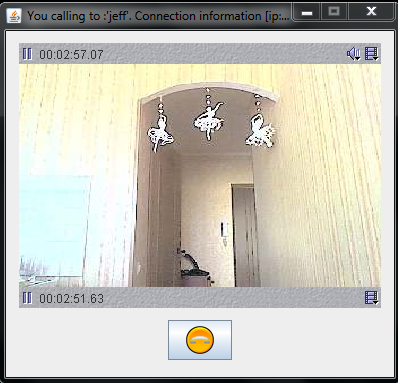
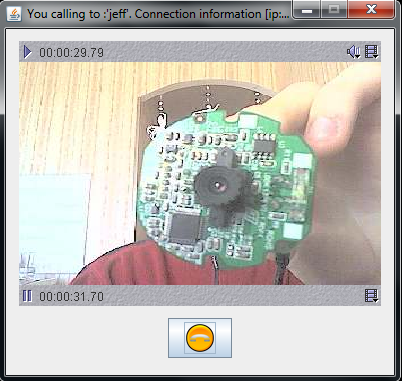
 

Рис. 14 – Видео чат

Для выбора устройства можно воспользоваться пунктом меню «Tools->Settings->Device».

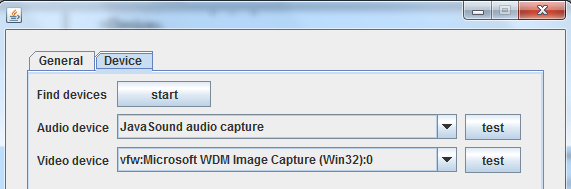


Рис. 15 – Выбор нужного устройства

Пользовательский интерфейс полностью отвечает всем требованиям, снащён сообщениями об ошибках и других событий. Хотелось бы заметить, что интерфейс полностью интуитивно понятен без всяких дополнительных документаций.

## 3.2 Сопроводительная документация программного обеспечения

Документация является неотъемлемой частью разработки программного продукта. Существует четыре основных типа документации на ПО:

1. архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО
2. техническая — документация на код, алгоритмы, интерфейсы, API
3. пользовательская — руководства для конечных пользователей, администраторов системы и другого персонала
4. маркетинговая

#### **Разработка описания программы**

Разработка описания приложения является неотъемлемым процессом любой разработки. Документация должна предоставлять полную информацию об использовании, поддержке и сопровождении. Каждый цикл разработки ПО должен документироваться. Это позволит контролировать процесс разработки, предоставляя тем самым механизм взаимодействия между разработчиками, безнес-аналистами, тестировщиками и заказчиком.

#### **3.2.2 Разработка руководства пользователя**

В отличие от технической документации, сфокусированной на коде и том, как он работает, пользовательская документация описывает лишь то, как использовать программу.

В случае если продуктом является программная библиотека, пользовательская документация и документация на код становятся очень близкими, почти эквивалентными понятиями. Но в общем случае, это не так.

Обычно, пользовательская документация представляет собой руководство пользователя, которое описывает каждую функцию программы, а также шаги, которые нужно выполнить для использования этой функции. Хорошая пользовательская документация идёт ещё дальше и предоставляет инструкции о том что делать в случае возникновения проблем. Очень важно, чтобы документация не вводила в заблуждение и была актуальной. Руководство должно иметь чёткую структуру; очень полезно, если имеется сквозной предметный указатель. Логическая связность и простота также имеют большое значение.

#### **3.2.3 Разработка технической документации**

При создании программы, одного лишь кода, как правило, недостаточно. Должен быть предоставлен некоторый текст, описывающий различные аспекты того, что именно делает код. Такая документация часто включается непосредственно в исходный код или предоставляется вместе с ним.

Подобная документация имеет сильно выраженный технический характер и в основном используется для определения и описания API, структур данных и алгоритмов.

Часто при составлении технической документации используются автоматизированные средства — генераторы документации, такие как Doxygen, javadoc, NDoc и другие. Они получают информацию из специальным образом оформленных комментариев в исходном коде, и создают справочные руководства в каком-либо формате, например, в виде текста или HTML.

Использование генераторов документации и документирующих комментариев многими программистами признаётся удобным средством, по различным причинам. В частности, при таком подходе документация является частью исходного кода, и одни и те же инструменты могут использоваться для сборки программы и одновременной сборки документации к ней. Это также упрощает поддержку документации в актуальном состоянии.

## 3.3 Анализ программного обеспечения

Для анализа программного обеспечения существует множество способов и подходов. Основными целями для анализа, являются качество кода и производительность.

#### **3.3.1 Анализ исходного кода приложения**

Для анализа исходного кода существует множество утилит, таких как Checksyle, PMD и т.д. Для проверки качества кода приложения использовались Checksyle и PMD. Checksyle очень удобная утилита, которая позволяет настраивать параметры качества кода. Если при сборке приложения будут обнаружены какие либо ошибки сhecksyle, то приложение не соберётся и будет сформирован специальный отчет с описанием всех ошибок. PMD работает аналогично Checksyle, разница только в области применения. PMD проверяет код на логические ошибки, а сhecksyle на ошибки оформления кода. Приложение «Видео чат» полностью отвечает сhecksyle и PMD.

#### **3.3.2 Анализ производительности приложения**

Для анализа производительности приложения используются JMeter, JProfiler, Perf4j и т.д. JMeter позволяет создавать специальные сценарии и запускать их. Так же можно настроить количество пользователей выполняющих сценарий и скорость возрастания нагрузки. Ниже на рисунке изображен отчет нагрузочного теста.

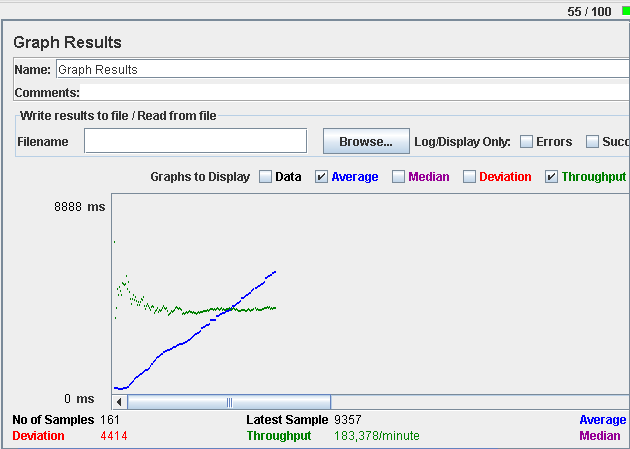


Рис. 15 – Результат нагрузочного теста

Number of Threads (users): 100, Ramp-Up Period (in seconds): 100. Эти настройки обозначают, что производительность будет тестироваться для ста пользователей, которые входят в систему по одному каждую секунду.

В правом верхнем углу можно наблюдать текущее количество виртуальных пользователей. Среднее время отклика (Average) растет, а скорость обработки (Throughput) не меняется. Это значит, что где-то на сервере операции становятся в очередь, и производительности не хватает, чтобы обслужить все запросы.

После выполнения всех нагрузочных тестов, можно отметить что приложение полностью масштабируемо, как горизонтально, так и вертикально.

## 3.4 Тестирование программного обеспечения

Существующие на сегодняшний день методы тестирования ПО не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого ПО.

Такой процесс формальной проверки или верификации может доказать, что дефекты отсутствуют с точки зрения используемого метода. (То есть нет никакой возможности точно установить или гарантировать отсутствие дефектов в программном продукте с учётом человеческого фактора, присутствующего на всех этапах жизненного цикла ПО).

Существует множество подходов к решению задачи тестирования и верификации ПО, но эффективное тестирование сложных программных продуктов — это процесс в высшей степени творческий, не сводящийся к следованию строгим и чётким процедурам или созданию таковых.

С точки зрения ISO 9126, Качество (программных средств) можно определить как совокупную характеристику исследуемого ПО с учётом следующих составляющих:

* 1. Надёжность
  2. Сопровождаемость
  3. Практичность
  4. Эффективность
  5. Мобильность
  6. Функциональность

Для определения качества программного продукта существует несколько разновидностей тестирования и тестов.

#### **3.4.1 Разработка модульных тестов**

Модульное тестирование, или юнит-тестирование (англ. unit testing) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель модульного тестирования — изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

Этот тип тестирования обычно выполняется программистами.

Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируются отдельные части программы, затем программа в целом.

Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду, минимизируя зависимости в системе.

Для создания unit-тестов использовались JUnit и EasyMock. Ниже приведены отчеты тестирования.

#### **3.4.2 Разработка методики интеграционных тестов**

Интеграцио́нное тести́рование (англ. Integration testing, иногда называется англ. Integration and Testing, аббревиатура англ. I&T) — одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «чёрного ящика».

#### **3.4.3 Результаты интеграционного тестирования**

Ниже изображен отчет о выполнении интеграционных тестов.

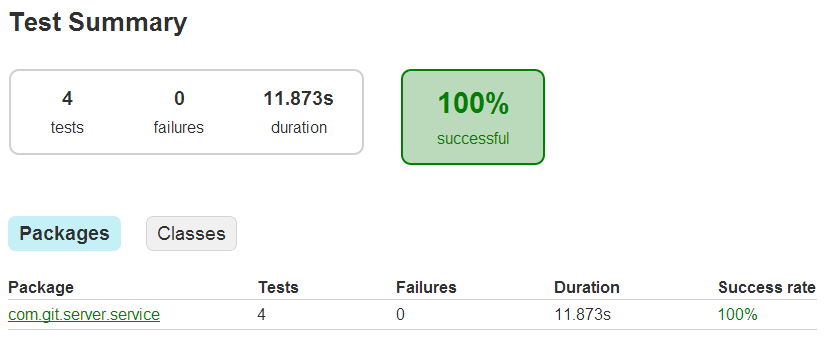


Рис. 16 – Результат интеграционных тестов

**Заключение**

В результате выполнения курсового проекта была разработано приложение «Видео чат».

Приложение написано на языке Java и по этому может запускаться на любой ОС где будет установлено JVM. В процессе реализации было придумано название для приложения «PhantomBox». PhantomBox удобное приложение позволяющее совершать видео вызовы и участвовать в видео конференциях. Стоит отметить, что данное приложение вызвало интерес у разработчиков OpenSource и вполне возможно что в будущем приложение будет развиваться и улучшаться.

При выполнении работы были получены новые знания и навыки работы с JMF.

Использование данного приложения не требует особых знаний и усилий со стороны пользователя и больших затрат ресурсов компьютера. Поэтому она может использоваться широким кругом пользователей.

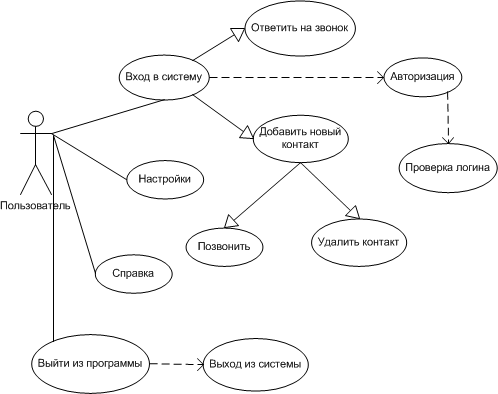
**Литература**

1. Кей С. Хорстманн. Java 2. Библиотека профессионала. "Вильямс", 2007.
2. Джошуа Блох. Effective Java. Programming Language Guide. "Лори" 2002.
3. Ричард Хелм, Ральф Джонсон, Джон Влиссидс.

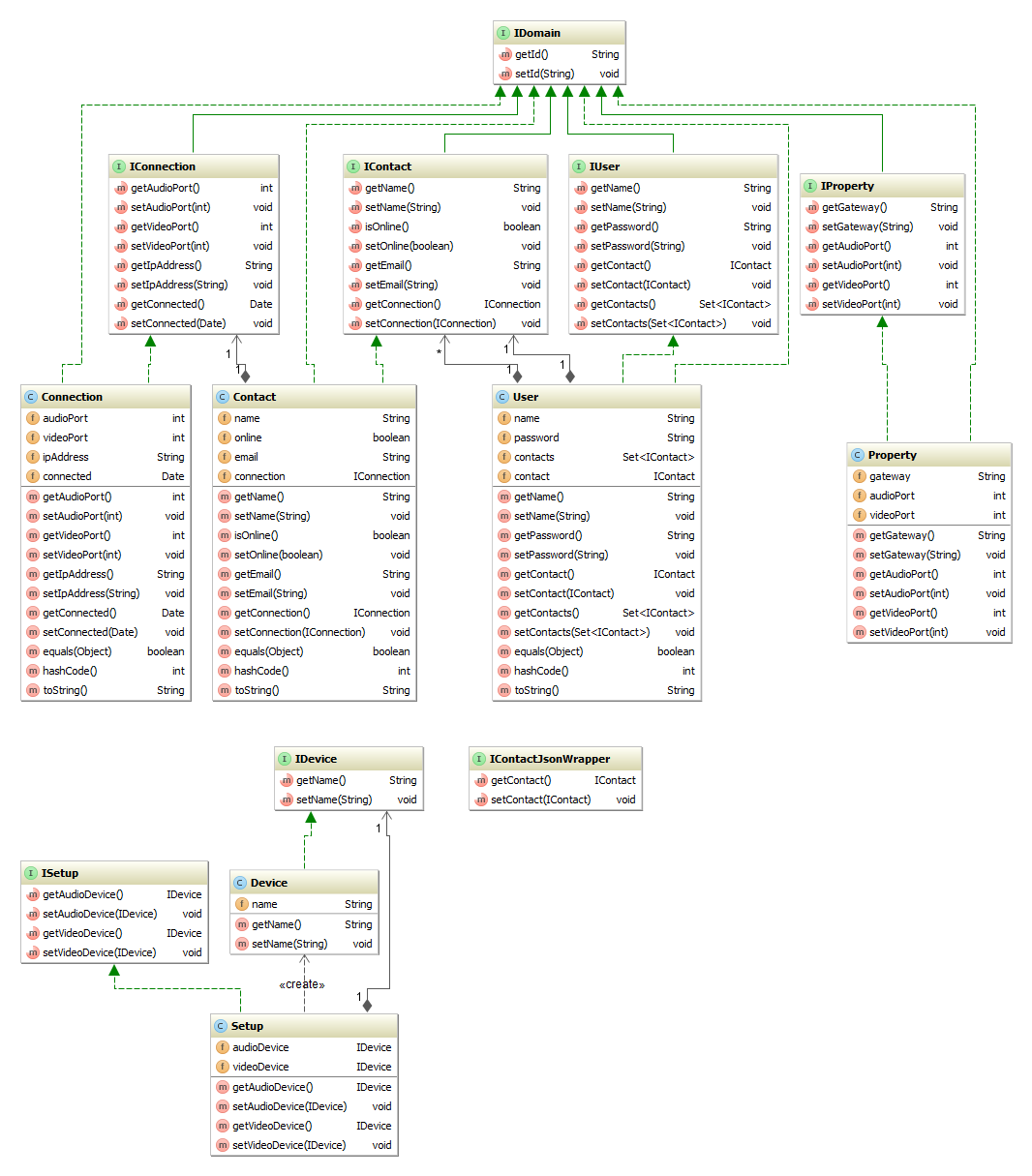
Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley 1994

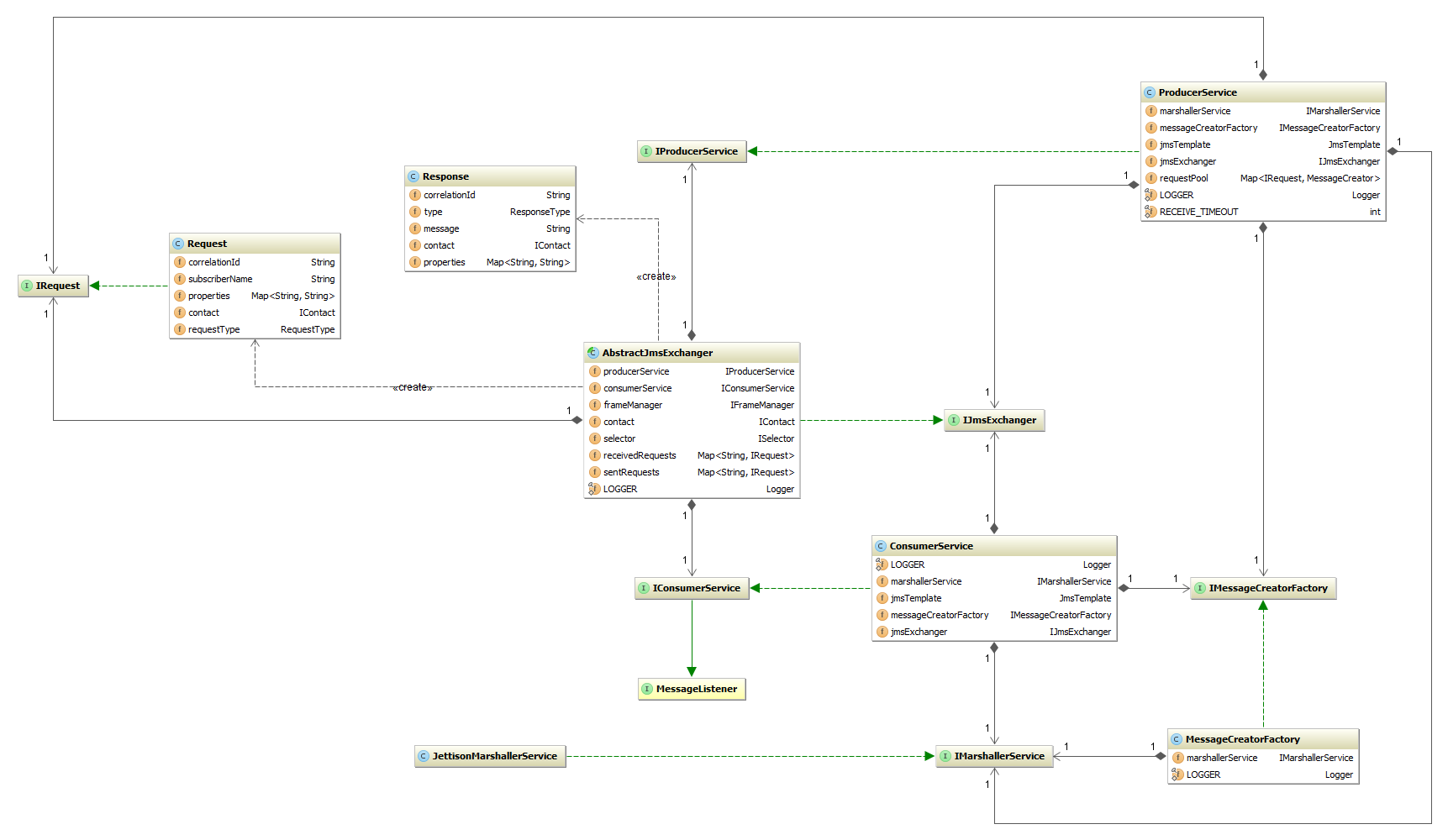
# **5. Перечень графического материала (с конкретным обозначением обязательных чертежей и графиков)**

# **5.1 Диаграмма вариантов использования**

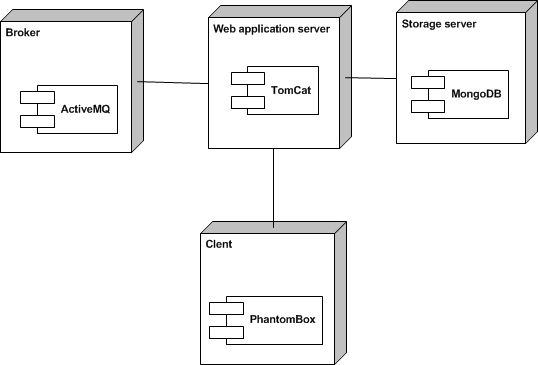


# **5.2 Диаграмма классов**

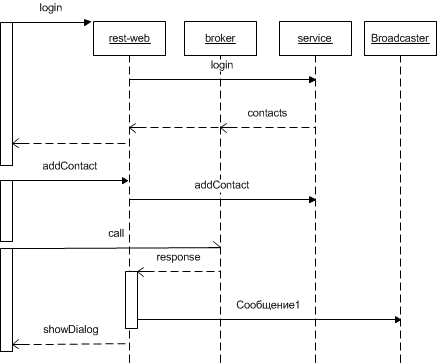




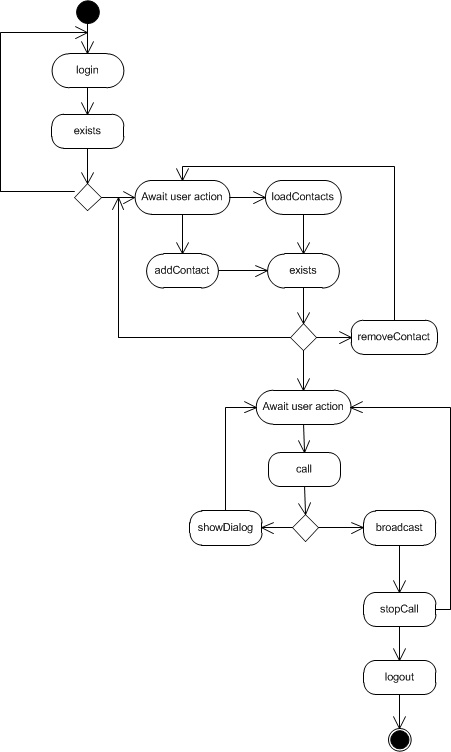
# **Диаграмма компонентов**



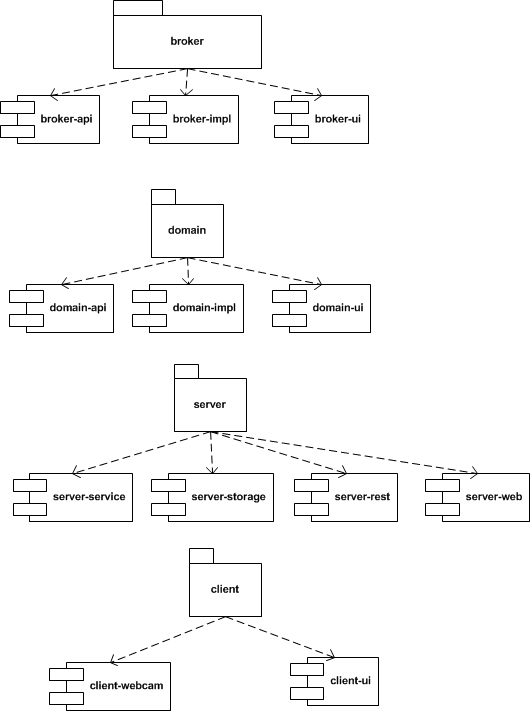
# **5.4 Диаграмма последовательности**



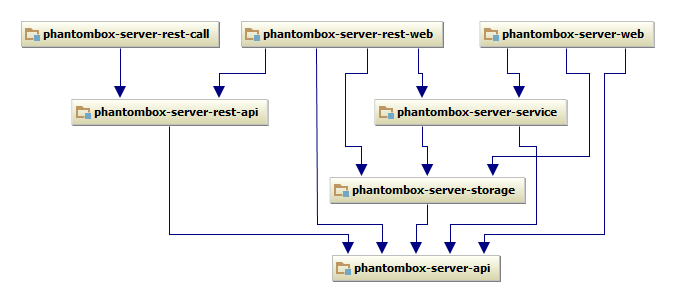
# **5.5 Диаграмма деятельности**



**5.6 Диаграмма логических уровней программы.**



**5.7 Диаграмма зависимостей**

.

