# ВВЕДЕНИЕ

Телекоммуникации играют очень важную роль в современном мире. Развитие технологий позволяет людям с разных уголков планеты общаться друг с другом, слышать голос и видеть друг друга в реальном времени. Мобильные операторы по всему миру предоставляют пользователям всё новые и новые сервисы, а качество обслуживания и площадь покрытия растут. Сейчас мы не можем представить свою жизнь без всех этих возможностей, и для современного пользователя становится недостаточно просто иметь возможность позвонить со своего мобильного телефона на телефон другого пользователя. С развитием технологий, растут и запросы пользователя.

Сейчас операторы предоставляют обилие всевозможных сервисов, таких как голосовая почта, удержание вызова, конференции, видео сообщения, мобильный интернет и др.

Изначально телекоммуникации строились на сетях с коммутацией каналов, что усложняло создание сложных сервисов и уменьшало общую производительность сетей. Однако с появлением сетей с коммутацией пакетов операторы всерьез задумались об изменении существующих стандартов. Разработкой взаимодействия телекоммуникационных сетей с технологиями пакетной коммутацией занялась группа [3GPP](https://ru.wikipedia.org/wiki/3GPP). Что вылилось в разработку спецификации IMS (IP Multimedia Subsystem).

IMS позволяет предоставлять услуги пользователям не зависимо от способа их доступа к сети, будь то мобильный телефон, стационарный телефон, или компьютер, путем организации системы шлюзов, запросы проходят через IMS, расположенную в сети с коммутацией пакетов и выходят через другие шлюзы, чтобы достигнуть конечного пользователя.

То, что IMS базируется на сетях с коммутацией пакетов, позволяет операторам, использующим эту спецификацию, выйти на новый уровень предоставления услуг. Горизонтальная архитектура IMS гарантирует легкую расширяемость, а возможности сетей с коммутацией пакетов значительно упрощают создание новых сервисов и повышают качество услуг. Так же, спецификация предполагает высокую модульность, что позволяет операторам без особых затрат (в сравнении с прошлыми технологиями) заменять отдельные компоненты системы.

Высокая модульность данной спецификации позволяет разрабатывать независимые сервисы, позволяя операторам комбинировать элементы сети от разных поставщиков, что дает разработчикам возможность сконцентрироваться в большей степени на реализации поставленной задачи, чем на организации взаимодействия сервисов между собой.

Цель данной дипломной работы – разработать сервер приложений сети IMS, реализующий функцию: “Гибкое перенаправление вызовов”.

Данная функция даст пользователям возможность управлять входящими звонками таким образом, чтобы вызов с большей вероятностью был отвечен, и разговор состоялся, возможно даже с другим пользователем (например с секретарем).

Использование спецификации Sip Servlets, позволит приложению быть развернутым на любом сервере, поддерживающем данный Java EE стандарт, что в свете распространённости технологии Java даёт ощутимое преимущество.

# 1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ И ПОСТАНОВКА ЗАДАЧИ

**1.1 Обзор предметной области**

Задачей дипломного проектирования ставится разработка SIP-сервера с поддержкой функции гибкого перенаправления вызовов.

SIP расшифровывается как Session Initiation Protocol - протокол инициирования сеанса, это протокол, разработанный IETF для VOIP (Voice Over IP) и для других сеансов передачи текста или мультимедиа данных, например, таких как, системы обмена мгновенными сообщениями, видео, игры в реальном времени и другие сервисы.

На протоколе SIP базируется подсистема IMS (IP Multimedia Subsystem), но так же содержит специфические расширения для телефонной связи, относящиеся, в частности, к обеспечению качества предоставления услуг (QoS), масштабируемости, аутентификации и биллингу.

В основе концепции IMS лежит передача через IP-уровень и сигнального, и пользовательского трафика и выполнение функций маршрутизации и управления сессиями абонентов с использованием информации об их состоянии. Тип сети доступа в концепции IMS не специфицируется – это может быть и сеть GSM, WCDMA или CDMA2000, и кабельная сеть широкополосного доступа, и WLAN.

Поэтому ожидается, что подсистема IMS станет оптимальным решением для предоставления мультимедийных услуг операторами стационарных и мобильных сетей. Пользователю подсистема IMS обеспечит связь как с другим пользователем, так и с контент-ресурсом в различных режимах (включая передачу голоса, текста, изображений и видео или любую их комбинацию) при максимальной персонализации услуг.

Под IP-телефонией подразумевается голосовая связь, которая осуществляется по сетям передачи данных, в частности по IP-сетям (IP — Internet Protocol). На сегодняшний день IP-телефония все больше вытесняет традиционные телефонные сети за счет легкости развертывания, низкой стоимости звонка, простоты конфигурирования, высокого качества связи и сравнительной безопасности соединения.

В традиционной телефонии установка соединения происходит при помощи телефонной станции и преследует исключительно цель разговора. Здесь голосовые сигналы передаются по телефонным линиям, через выделенное подключение. В случае же IP-телефонии, сжатые пакеты данных поступают в глобальную или локальную сеть с определенным адресом и передаются на основе данного адреса. При этом используется уже IP-адресация, со всеми присущими ей особенностями (такими как маршрутизация).

При всех достоинствах IP-телефонии, она так же является более выгодным решением в финансовом плане, как для оператора так и для абонента, по ряду причин:

* Сети с коммутацией пакетов обладают избыточным запасом производительности, не используя все возможности сети на полную мощность, в то время, как IP-телефония использует технологию пакетной передачи данных, позволяя полностью задействовать пропускную способность сети.
* Доступ в глобальную сеть на сегодняшний момент есть почти у всех, это позволяет значительно снизить затраты на подключение абонентов.
* В локальной сети можно с легкостью развернуть внутренний сервер, что позволит не использовать внешнюю АТС.
* IP-телефония позволяет улучшить качество связи. Достигается это, опять же, благодаря трем основным факторам:
  + Телефонные серверы постоянно совершенствуются и алгоритмы их работы становятся более устойчивыми к задержкам или другим проблемам IP-сетей.
  + В частных сетях их владельцы обладают полным контролем над ситуацией и могут изменять такие параметры, как ширина полосы пропускания, количество абонентов на одной линии, и, как следствие, величину задержки.
  + Сети с коммутацией пакетов развиваются, и ежегодно вводятся новые протоколы и технологии, позволяющие улучшить качество связи.

Система IMS использует множество протоколов и описывает большой набор элементов. Это позволяет избежать стандартизации сервисов, вместо этого стандартизуются интерфейсы взаимодействия элементов системы IMS, что значительно ускоряет создание новых сервисов. И для полноценного понимания концепции IMS необходимо понимать используемые протоколы, и принципы взаимодействия между элементами системы.

**Основные протоколы используемые в IMS**:

*SIP*

SIP (Session Initiation Protocol) – протокол сигнализации, предназначенный для организации, изменения и завершения сеансов связи. SIP независим от транспортных технологий, однако при установлении соединения предпочтительно использовать UDP. Для передачи самой голосовой и видеоинформации рекомендовано применять RTP, но возможность использования других протоколов не исключена. В SIP определены два типа сигнальных сообщений — запрос и ответ. Разработчики протокола сип позаимствовали основные принципы у протоколов SMTP (Simple Mail Transfer Protocol) и HTTP (Hipertext Transfer Protocol).

Существует шесть основных процедур:

* INVITE (приглашение) — приглашает пользователя принять участие в сеансе связи (служит для установления нового соединения; может содержать параметры для согласования);
* BYE (разъединение) — завершает соединение между двумя пользователями;
* OPTIONS (опции) — используется для передачи информации о поддерживаемых характеристиках (эта передача может осуществляться напрямую между двумя агентами пользователей или через сервер SIP);
* АСК (подтверждение) — используется для подтверждения получения сообщения или для положительного ответа на команду INVITE ;
* CANCEL (отмена) — прекращает поиск пользователя;
* REGISTER (регистрация) — передает информацию о местоположении пользователя на сервер SIP, который может транслировать ее на сервер адресов (Location Server).

Выражаясь простым языком: протокол SIP контролирует звонок. Большинство элементов системы IMS, предоставляющих непосредственные услуги конечному пользователю, взаимодействует именно по этому протоколу. Пользовательские устройства либо сами генерируют и обрабатывают SIP сообщения, (например в случае если взаимодействие происходит через интернет, с использованием специального ПО), или же их запросы прежде чем попасть в IMS сеть проходят через специальные шлюзы, как в случае телефонных сетей общего пользования. Далее сообщения могут пройти через ряд прокси серверов, и серверов предоставления услуг, прежде чем попадут в точку назначения. Опять же на выходе из сети IMS сообщение может попасть к конечному пользователю, проходить через ряд шлюзов в другие сети, попасть в другую IMS сеть и так далее.

*Diameter*

Diameter – был выбран в качестве ААА (Аутентификация Авторизация и Аккаунтинг(учет)) протокола в сети IMS. Diameter это эволюция протокола Radius, который обширно использовался в интернете для целей ААА.

Diameter состоит из базового протокола, который расширяется так называемыми Diameter приложениями (Diameter Applications). Diameter приложения – это настройки или расширения Diameter протокола служащие определенной цели в конкретно заданной среде.

SIP серверы используют Diameter для получения и изменения информации о пользователе, подписываются на события, связанные с изменением определенных полей пользовательской информации и многое другое.

В дополнение к вышеперечисленным протоколам SIP и Diameter существует множество дополнительных протоколов, таких как **H.248**, используемый для управления медиа шлюзами, **RTP** (Real Time Transfer Protocol) и **RTCP** (RTP Control Protocol), используемые для передачи медиа данных в режиме реального времени, таких как аудио и видео данные. И другие.

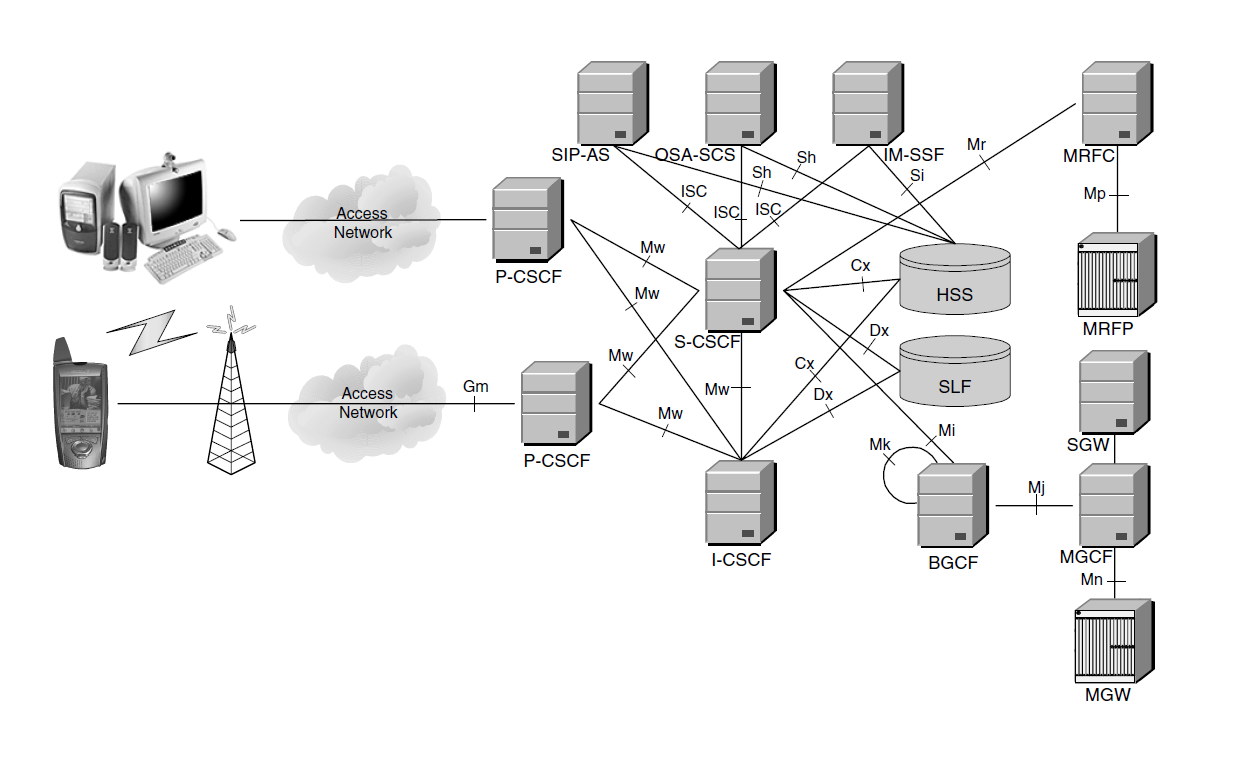


Рисунок 1.1 – Архитектура IMS

**Основные элементы IMS сети и их краткое описание.**

*Базы данных: HSSи SLF*

HSS(Home Subscriber Server) – это центральное хранилище информации о пользователях. Он содержит всю пользовательскую информацию, необходимую для установления мультимедийной сессии. Эта информация включает помимо прочего, информацию о местоположении, информацию безопасности, информацию о профиле пользователя.

В сетях с большим количеством пользователей один HSS может не справиться со всеми запросами, для этих целей необходим SLF(Subscription Locator Function). SLF просто связывает пользователя с конкретным HSS.

HSS, как и SLF реализуют протокол Diameter.

*CSCF*

CSCF(Call/Session Control Function) – это SIP сервер контролирующий звонок/сессию и является важнейшим узлом IMS сети. CSCF осуществляет менеджмент SIP запросов в IMS.

Существует 3 типа CSCF:

1. Proxy CSCF (P-CSCF) – это первая точка контакта между терминалом конечного пользователя и IMS сетью.
2. Interrogating CSCF (I-CSCF) – это узел, служащий для приема сообщений из других IMS сетей.
3. Serving CSCF (S-CSCF) – этот элемент выполняет управление SIP запросами внутри IMS сети, отправляет запросы на серверы приложений (Application Server) и имеет доступ к пользовательской информации.

*SIP AS*

SIP AS (Application Server) – это сервер приложений в сети IMS, который выполняет IP мультимедиа услуги на базе протокола SIP. Узел данного типа будет разработан в результате выполнения данного дипломного проекта. Ниже будут перечислены основные разновидности SIP серверов приложений.

*Прокси Сервер* – компонент, выполняющий посреднические функции, который выступает в роли как сервера, так и клиента, в целях создания запросов от лица других клиентов. Основная роль, которую играет прокси сервер - это маршрутизация, имеется ввиду, что он выполняет работу по гарантированной отправке запросов другому участнику, который "недоступен" для абонента отправившего запрос. Еще одно применение прокси серверов - это ограничение доступа к сервисам (например, он может проверять, имеет ли пользователь право совершать вызовы). Прокси сервер сам обрабатывает, и, при необходимости, вносит изменения в определенные поля сообщений запросов, перед тем как переслать их.

SIP прокси сервер - это участник на пути маршрутизации SIP запросов к серверу пользователя (user agent servers) и доставки SIP ответов обратно к пользовательскому агенту (user agent clients). Запрос может пройти через несколько прокси серверов на своем пути, до того как он достигнет UAS. Каждый из них принимает решение по маршрутизации запроса, вносит необходимые изменения в сообщение и пересылает его к следующему элементу на пути маршрута SIP сообщения. Ответные сообщения возвращаются тем же путем, что и запрос, через те же прокси сервера, но в обратной последовательности.

Предусмотрено два типа прокси-серверов

* с сохранением состояний (stateful). Такой сервер хранит в своей памяти все полученные запросы и связанные с ним новые сформированные запросы до окончания транзакции.
* без сохранения состояний(stateless). Такой сервер просто обрабатывает получаемые запросы. Но на его базе нельзя реализовать сложные, интеллектуальные услуги.

*Redirect Server (Сервер переадресации)* – используется для перенаправления вызова по адресу текущего местоположения пользователя. Сервер переадресации не терминирует вызовы и не инициирует собственные запросы, а только сообщает адрес необходимого терминала или прокси-сервера при помощи ответов класса 3XX (301 Moved Permanently или 302 Moved Temporarily).

*Registrar (Сервер регистрации)*

Протокол SIP подразумевает мобильность пользователя, т.е. пользователь может перемещаться в пределах сети, получая новый адрес. Поэтому в SIP существует механизм регистрации - уведомление о новом адресе со стороны пользовательского агента. Сервер регистрации или регистратор служит для фиксации и хранения текущего адреса пользователя и представляет собой регулярно обновляемую базу данных адресной информации. В общем случае, пользователь сообщает серверу регистрации свою адресную информацию, такую как IP-адрес или доменное имя и абонентский телефонный номер - при помощи запроса REGISTER. Сервер может подтвердить успешную регистрацию (200 OK) или отклонить, в случае если есть проверка данных и учетная запись пользователя не найдена (404 Not found) или регистрация для пользователя запрещена в данный момент (403 Forbidden). Регистратор может указать на необходимость логина пользователя для проверки (401 Unauthorized), а также предложить цифровую аутентификацию на основе зашифрованного пароля.

*B2BUA* — (back-to-back user agent, буквально: пользовательский агент спина-к-спине) — вариант серверного логического элемента в приложениях, работающих с протоколом SIP. По идеологии работы, B2BUA похож на прокси-сервер SIP, однако есть принципиальные различия. Сервер B2BUA, работает одновременно с несколькими (как правило, двумя) конечными устройствами — терминалами, разделяя вызов или сеанс на разные плечи-участки. С каждым участком B2BUA работает индивидуально, как UAS по отношению к инициатору и как UAC по отношению к терминалу, принимающему вызов. При этом сигнальные сообщения передаются в рамках сеанса в обе стороны синхронно, хотя решение о необходимости передачи сообщения и его формате принимается B2BUA для каждого участка в индивидуальном порядке. Каждый из участников соединения (сеанса связи), на уровне сигнализации взаимодействует с B2BUA, как с оконечным устройством, хотя в действительности, сервер является посредником. Это отражается в адресных полях (таких как From, To и Contact) сообщений, отправляемых сервером B2BUA. Таким образом, ключевое отличие B2BUA - полностью независимая сигнализация всех участков вызова. Это означает, в частности, что для взаимодействия с каждым отдельным пользователем в рамках сеанса связи используются уникальные идентификаторы, а содержимое одних и тех же сообщений для разных участков будет различным. Пользовательские агенты оконечных терминалов могут взаимодействовать с B2BUA и при участии прокси-серверов.

Сервер B2BUA может предоставлять следующие функции:

* Управление звонками (биллинг, перевод звонка, автоматическое разъединение и т. д.)
* Сопряжение разных сетей (в частности, для адаптации разных диалектов протокола, зависимых от производителей)
* Сокрытие структуры сети (частные адреса, сетевая топология и т. п.)

## 1.2 Функции и задачи объекта автоматизации

Объектом автоматизации данного дипломного проекта является SIP сервер приложений, а именно его разновидность B2BUA, описанная в предыдущем подразделе.

Основной функцией SIP сервера приложений является установка сеанса мультимедиа связи между двумя конечными пользователями (или более, в случае конференции), и управление этим сеансом.

В задачи SIP сервера приложений входит:

* Принимать SIP запросы от пользователей и других серверов.
* Отправлять SIP запросы пользователям и другим серверам.
* Принимать SIP ответы от пользователей и других серверов.
* Отправлять SIP ответы пользователям и другим серверам.
* Уметь поддерживать сеанс связи на базе протокола SIP.
* Получать информацию о пользователе и модифицировать ее.
* Предоставлять конечному пользователю мультимедийные услуги на базе протокола SIP.
* Извлекать настройки пользователя, в определенном формате, и правильно их интерпретировать.

## 1.3 Определение целесообразности автоматизации

На текущий момент на рынке существует порядка достаточно много SIP серверов приложений, начиная бесплатными решениями, предоставляющими только базовый функционал, заканчивая крупными коммерческими системами, реализующими не только SIP сервер приложений но и многие другие элементы IMS сети.

**Основные решения на рынке SIP серверов**

*Asterisk*

Asterisk — бесплатная программная АТС, способная коммутировать как VoIP вызовы, так и вызовы, осуществляемые между IP-телефонами и традиционной телефонной сетью общего пользования.

*Cisco Unified Communication Manager (CallManager).*

CallManager предназначен скорее для крупных сетей, включающих до 30000 абонентов. Данный программно-аппаратный комплекс обеспечивает надежность работы и позволяет конфигурировать множество параметров, таких как переадресация звонков или голосовое меню. Существует и “облегченная” express версия, предназначенная скорее для небольших офисов.

Из преимуществ Cisco CallManager следует отметить в первую очередь знаменитую техническую поддержку корпорации Cisco. При соответствующем уровне контракта на обслуживание, любая проблема, начиная с вопросов по настройке и заканчивая вышедшим из строя оборудованием, будет решена практически мгновенно. Поэтому Cisco CallManager подойдет компаниям, готовым платить немалые деньги, но и получать при этом высочайшее качество обслуживания.

*Avaya IP Office*

Система IP Office может стать неплохим выбором для среднего размера телефонной сети. Количество абонентов здесь ограничено не только мощностью сервера, но и количеством приобретенных лицензий. Лицензировать необходимо практически все — платы расширения, используемые приложения и т.д., что может доставить определенные неудобства.

Видя обилие готовых решений можно сделать вывод, что нецелесообразно создавать новое приложение, особенно учитывая, что для создания чего то серьезного необходима большая команда высоко квалифицированных разработчиков и немалые временные затраты.

Однако существует спецификация Java Sip Servlets, которая позволяет писать независимые приложения и комбинировать их для предоставления услуг в сети IMS.

Каждое отдельное приложение может быть создано отдельно, и использовано множеством различных поставщиков, с небольшими затратами на ввод в эксплуатацию, при условии что их IMS сеть базируется на спецификации Java Sip Servlets.

Так же данная спецификация описывает взаимодействие Sip сервлетов с общеизвестными HTTP сервлетами в рамках одного приложения, что позволяет инициировать SIP диалог прямо из окна браузера. А широкая распространенность языка Java в современном мире информационных технологий дает большие возможности для развития и распространения данного решения.

Целью автоматизации является предоставление пользователям некоторой IMS сети возможности гибко настраивать перенаправление входящих вызовов

Разрабатываемое в рамках дипломного проектирования ПО позволит конечным пользователям быть доступными для звонка в разных местах в зависимости от различных условий, указанных ими в настройках, и обеспечит:

1. максимальное удобство для пользователя;
2. большую привлекательность среди пользователей для поставщика услуги;

Данный сервис позволит пользователю не пропустить важные звонки, или же перенаправить их на ответственных людей в зависимости от времени суток, и дня недели. Например в рабочие дни он может сконфигурировать, что если он недоступен, чтобы звонки поступали к нему в офис, затем к его секретарю, и в конце в комнату для конференций, таким образом с большей вероятностью важный звонок не будет пропущен.

Условиями для запуска перенаправления вызовов могут быть недоступность пользователя, занятость линии, или отсутствие ответа, в течении определенного периода. Что в совокупности с возможностью указать временные промежутки выполнения сервиса, и списки целей, которые будут вызываться в последовательном, параллельном или смешанном режимах, и дают сервису гибкость, упомянутую в его названии.

## 1.4 Разработка вариантов автоматизации и выбор оптимального варианта

Разрабатываемое ПО будет состоять из SIP сервера приложений и базы данных. В базе данных будет храниться информация о пользователях, их текущем положении (IP-адрес и порт), и конфигурации сервиса.

Данные, хранящиеся в базе данных, могут быть описаны следующими моделями представления:

* иерархическая;
* сетевая;
* реляционная.

Эти 3 модели являются наиболее распространенными при разработке, однако в последнее время набирают популярность такие модели как: пост-реляционная, объектно-ориентированная, многомерная, объектно-реляционная и другие.

В дипломном проекте будет использоваться реляционная модель. Преимуществами этой модели можно считать отображение информации в наиболее простой для пользователей форме. Так же данная модель основана на развитом математическом аппарате, который позволяет достаточно лаконично описывать основные операции над данными.

Выбор архитектуры приложения очевиден, приложение по своей сути является SIP сервером, к которому будут отправлять запросы клиентские устройства (SIP- софтфоны). Сам же сип сервер не является типичным сервером, так как он может как выполнять запросы клиентов, так и сам создавать запросы, играя роль клиента, то есть SIP сервер содержит в себе как серверный функционал, так и клиентский.

Сервером называется компьютер либо программа, управляющий ресурсом. Клиентом – компьютер (программа), использующий ресурс.

В разделе 1.1 были перечислены основные виды SIP серверов. Сервисные функции могут предоставлять в основном Прокси сервера, и B2BUA сервера.

Так как B2BUA сервера по спецификации SIP имеют больше возможностей в плане управления сеансом связи, то в рамках данного дипломного проекта будет реализован именно B2BUA.

Схема работы B2BUA представлена на рисунке 2.1.

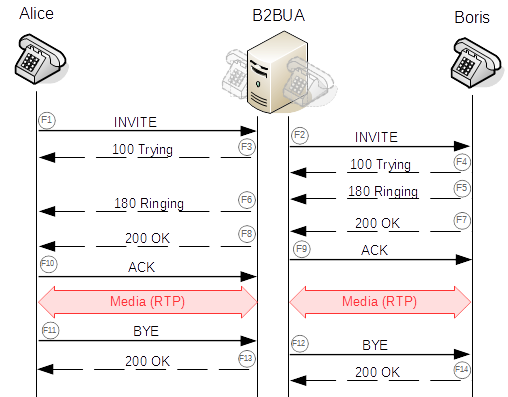


Рисунок 1.2 – Схема работы B2BUA

После выбора модели представления данных необходимо позаботиться о выборе конкретной СУБД

СУБД (Система управления базами данных) – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования баз данных многими пользователями.

СУБД позволяет вставлять, обновлять, удалять и извлекать информацию из базы данных, что обычно осуществляется с помощью языка манипулирования данными (DML — Data Manipulation Language). Наличие централизованного хранилища всех данных и их описаний позволяет использовать язык DML как общий инструмент организации запросов, который иногда называют языком запросов (query language).

В реляционных базах данных используется язык SQL (Structured Query Language) или, по-русски, Структурированный Язык Запросов. SQL является саммым распространенным на сегодняшний момент языком манипулирования данными и существует множество реляционных СУБД, поддерживающих данный язык, фактически он является обязательным для реляционной СУБД.

Реляционные базы данных манипулируют таблицами, и связями между ними, так же существуют и другие сущности, подробнее о которых можно узнать из открытых источников информации.

Язык SQL позволяет создавать, изменять и удалять таблицы, так же как и запрашивать информацию из таблиц, поддерживая вложенные запросы и условные операторы.

СУБД для данного дипломного проекта должна быть открытой и легковесной, при этом должна предоставлять все основные возможности языка SQL и иметь хорошую документацию.

Под эти критерии отлично подходит открытая СУБД PostgreSQL.

Сильными сторонами PostgreSQL считаются:

* производительные и надёжные механизмы [транзакций](https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D0%B8%D1%8F_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) и [репликации](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BF%D0%BB%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_(%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0));
* расширяемая система встроенных языков программирования;
* [наследование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5));
* легкая расширяемость.

## 1.5 Постановка задачи на создание системы

В рамках данного дипломного проекта будет разработан SIP сервер приложений, с функцией гибкого перенаправления вызовов. В состав сервера будет так же входит реляционная база данных под управлением СУБД PostgresSQL.

В функции сервера будет входить:

* обработка запросов по протоколу SIP в соответствии со спецификацией.
* установление сеанса связи по протоколу SIP между двумя клиентами.
* гибкое перенаправление вызовов на основе настроек пользователя из БД.
* регистрация пользователя в сети IMS.
* организация многопоточной модели обработки и отправления запросов.
* обработка таймеров.
* настройка медиа сессии по протоколу SDP.

Эксплуатационные характеристики для системы следующие:

* регистрация может осуществляться только для пользователей присутствующих в БД.
* времена отклика на запрос не должно превышать 1 секунду;
* время восстановления БД не должно превышать одни сутки;
* операционная система WINDOWS 95/98/NT/2000/xp/7;
* наличие локального сервера Apache Tomcat/JBoss.
* размер необходимой оперативной памяти 1024 Мб и более;
* процессор не ниже intel core i3 (3,2 Ghz);

# 2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

## 2.1 Структура системы

Система, это совокупность взаимосвязанных элементов, которые функционируя вместе выполняют определенную задачу, и находятся в связи друг с другом так, что эту совокупность можно рассматривать как единое целое.

Программная система, это система, состоящая из программного и аппаратного обеспечения и данных.

На рисунке 2.1 изображена общая структура разрабатываемой системы.

SIP-клиент

SIP-клиент

IP-сеть

SIP-сервер

СУБД

B2BUA

Регистратор

База данных

Рисунок 2.1 – Структура системы

Разрабатываемая система имеет клиент-серверную архитектуру, схожую с архитектурой веб приложений, однако в отличие от веб приложений, SIP сервер может выступать как в качестве сервера, так и в качестве клиента, и это все за один сеанс связи. Еще одним отличием является то, что в веб приложениях разработчик обычно реализует как серверную, так и клиентскую часть, в виде HTML страниц и Java script кода, которые затем интерпретируются браузером. В данном проекте в качестве клиента может выступать любое устройство, поддерживающее протокол sip, в частности так называемые sip софтфоны (softphone).

Так же можно заметить что две логические сущности размещены в одном блоке: Регистратор и B2BUA. Реализовывать отдельный полнофункциональный регистратор в рамках данной дипломной работы не имеет смысла, так как подобного программного обеспечения уже хватает на рынке в открытом доступе. Основную логику приложения будет выполнять сущность B2BUA (Back to back user agent). Подробнее о данном типе SIP серверов было рассказано в разделе 1.1.

Регистрация необходима для тестирования функционала услуги *гибкого перенаправления вызовов*, однако полный функционал регистраторов является излишним. Поэтому вместо внедрения полнофункционального готового решения будет разработана логика регистрации в рамках общего SIP сервера. Разделение между функционалом регистрации и гибкого перенаправления вызовов, будет видно на уровне модулей программного кода, в случае языка Java – на уровне классов и интерфейсов.

Таким образом разрабатываемый SIP сервер будет действовать как регистратор, на запросы SIP REGISTER, выполняя, как регистрацию, так и де-регистрацию, что подразумевает занесение и удаление информации из базы данных. А в случае же других SIP запросов (например INVITE) сервер будет действовать как B2BUA, предоставляя доступ к функционалу *гибкого перенаправления вызовов*.

## 2.2 Структура программного обеспечения

Разрабатываемое ПО предназначено организации сеанса связи по протоколу SIP, с возможностью *гибкого перенаправления вызовов*

Взаимодействие пользователя с ПО осуществляется за счет клиентских SIP приложений. Данные будут храниться в реляционной базе данных, поддерживающей такие операции как выборка вставка и удаление.

Для разработки ПО выбран язык Java. Язык программирования Java – это многоцелевой, строго-типизированный, основанный на классах объектно-ориентированный язык. Программы написанные на языке Java являются кросс-платформенными, и могут запускаться на любом устройстве, при условии что для этого устройства существует Java виртуальная машина (JVM) обладающая всем необходимым функционалом.

Язык Java это популярный выбор при разработке серверных приложений, в частности веб-приложений. Так как язык Java обладает спецификацией Java servlets и ее реализацией Java HTTP Servlets.

Сервлет является интерфейсом Java реализация которого расширяет функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа запрос-ответ.

Данная спецификация позволяет разрабатывать веб-приложения на высоком уровне абстракции, не заботясь о взаимодействии по протоколу (например HTTP, SIP) а концентрируясь на логике работы приложения. Задачи низкого уровня выполняются контейнером сервлетов.

Контейнер сервлетов — это программа, представляющая собой [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)), который занимается системной поддержкой [сервлетов](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%BB%D0%B5%D1%82) и обеспечивает их жизненный цикл в соответствии с правилами, определёнными в спецификациях. Может работать как полноценный самостоятельный веб-сервер, быть поставщиком страниц для другого веб-сервера, например [Apache](https://ru.wikipedia.org/wiki/Apache_HTTP_Server), или интегрироваться в [JavaEE](https://ru.wikipedia.org/wiki/Java_EE) [сервер приложений](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B9). Обеспечивает обмен данными между сервлетом и клиентами, берёт на себя выполнение таких функций, как создание программной среды для функционирующего сервлета, идентификацию и авторизацию клиентов, организацию сессии для каждого из них.

В данном проекте будет использована спецификация Sip servlets.

Sip сервлеты, как и HTTP сервлеты, это блоки java кода хранимые и управляемые контейнером SIP сервлетов. Контейнер действует на уровне протокола, передавая полученные запросы определенному SIP сервлету, а сервлет уже в свою очередь, может создавать ответы на эти запросы, создавать новые запросы или работать как прокси. Внутри контейнера сервлетам предоставляется абстрактное, высокоуровневое видение протокола SIP через набор хорошо определенных интерфейсов, что позволяет разработчикам сконцентрироваться на бизнес логике приложения.

Создание Sip сервлета заключается в создании Java класса, наследующегося от класса javax.servlet.sip.SipServlet и перегрузке методов этого класса.

Класс SipServlet предоставляет удобный интерфейс позволяющий обрабатывать любое сообщение по протоколу SIP, будь то запрос или ответ, в отдельном методе с соответствующим названием. Например для обработки сообщения INVITE используется метод SipServlet.doInvite(), а для обработки ответа группы 2xx (например 200 Ok) используется метод SipServlet.doSuccessResponse()

В качестве параметров эти методы принимают объекты SipServletRequest или SipServletResponse в зависимости от того обрабатывает данный метод запрос, или ответ соответственно.

Контейнер сервлетов, при получении SIP сообщения, обрабатывает его, выполняя все необходимые действия менеджмента сессий и тому подобное, и затем вызывает метод сервлета SipServlet.service(), в который передаются два объекта SipServletRequest и SipServletResponse, один из которых будет равняться null. Таким образом в зависимости от того, какой объект был не равен null, и от параметров этого объекта и бует определено какой конкретный метод обработки вызвать.

Реализация сложной логики напрямую в методах сервлета является плохим примером программирования, и зачастую является просто не выполнимой задачей, так как код становится неструктурированным, запутанным и неподдерживаемым.

При разработке сложных приложений важно разработать удачную структуру программного обеспечения.

Приложение необходимо разбить на максимально независимые друг от друга модули, которые могут быть разработаны отдельно и заменены другими с течением времени.

Используя язык Java и его структуру пакетов удалось спроектировать модульную систему удовлетворяющую вышеизложенным требованиям.

В результате были выделены следующие модули, указанные в таблице 2.1.

Таблица 2.1 – Модули ПО

|  |  |
| --- | --- |
| Название | Описание |
| FlexibleComminicationServlet | Главный сервлет приложения, который использует другие модули для обработки полученных сообщений |
| config | Модуль предназначенный для обработки пользовательских настроек полученных из базы данных. |
| error | Этот модуль содержит специфические исключения (Exception) используемые в приложении. |
| util | Этот модуль предоставляет вспомогательный функционал, который может быть использован во всех модулей приложения. |
| service | Данный модуль отвечает за основной функционал приложения и содержит в себе следующие подмодули:   * database – модуль отвечающий за доступ к базе данных * registrar – модуль отвечающий за функционал регистрации. * fsm – модуль реализующий машину состояний сервиса *гибкое перенаправление вызовов* |

Результаты проектирования структуры ПО изображены на рисунке 2.1.

flexible-communtication

util

error

service

FlexibleCommunicationServlet

config

CommonUtils

SdpUtils

database

registrar

fsm

action

condition

rule

target

Рисунок 2.1 – Структурная схема ПО

В модуле **fsm** будет реализован основной функционал сервиса – *гибкое перенаправление вызовов.* Аббревиатура **fsm** расшифровывается как *finite-state machine*, что в переводе с английского означает конечный автомат.

Для реализации будет применен шаблон проектирования State (состояние).

Назначение паттерна State:

* Паттерн State позволяет объекту изменять свое поведение в зависимости от внутреннего состояния. Создается впечатление, что объект изменил свой класс.
* Паттерн State является объектно-ориентированной реализацией конечного автомата.

Поведение объекта зависит от его состояния и должно изменяться во время выполнения программы. Такую схему можно реализовать, применив множество условных операторов: на основе анализа текущего состояния объекта предпринимаются определенные действия. Однако при большом числе состояний условные операторы будут разбросаны по всему коду, и такую программу будет трудно поддерживать.

На рисунке 2.2 изображена UML-диаграмма классов шаблона State.

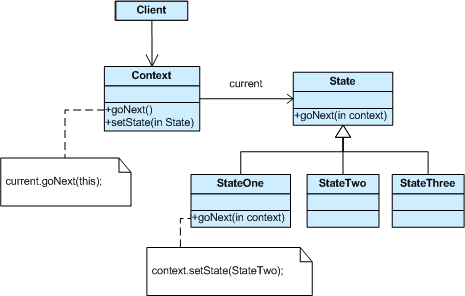


Рисунок 2.2 – UML-диаграмма классов шаблона State

В данном проекте будет реализован класс ServiceContext, который будет хранить указатель на текущее состояние ServiceState. Набор состояний будет уточнен в процессе разработки.

Полный список состояний и краткое их описание доступны в разделе номер 3 данной дипломной работы.

## 2.3 Структура информационного обеспечения

*Так как мало таблиц, большое внимание уделить настройкам и их формату(JSON)*

Информационное обеспечение данного проекта будет представлено реляционной базой данных.

На данном этапе необходимо спроектировать структуру базы данных что является неотъемлемой частью разработки программного обеспечения.

Проектирование БД принято разбивать на следующие этапы:

1. Концептуальное проектирование базы данных;
2. Логическое проектирование базы данных;
3. Физическое проектирование базы данных.

**Концептуальное проектирование**

Концептуальное проектирование - сбор, анализ и редактирование требований к данным. Для этого осуществляются следующие мероприятия:

* обследование предметной области, изучение ее информационной структуры;
* выявление всех фрагментов, каждый из которых характеризуется пользовательским представлением, информационными объектами и связями между ними, процессами над информационными объектами;
* моделирование и интеграция всех представлений.

По окончании данного этапа получаем концептуальную модель, инвариантную к структуре базы данных. Часто она представляется в виде модели "сущность-связь"[].

В ходе обследования предметной области выявляется перечень задач системы связанных с информационным обеспечением.

Основной перечень задач системы:

1. Добавление первичного пользователя.
2. Изменение (добавление, удаление) настроек первичного пользователя.
3. Регистрация конкретного устройства, с привязкой к первичному пользователю.
4. Де-регистрация устройства пользователя.
5. Получение настроек пользователя.
6. Определение, является ли пользователь первичным.
7. Определение, является ли пользователь зарегистрированным.
8. Получение списка зарегистрированных устройств первичного пользователя.

Настройки пользователя можно представить набором связанных таблиц, однако это нецелесообразно, так как значительно увеличит нагрузку на базу данных. Поэтому было принято решение в качестве настроек использовать формат JSON.

JSON (JavaScript Object Notation) - простой формат обмена данными, удобный для чтения и написания как человеком, так и компьютером. Он основан на подмножестве [языка программирования JavaScript](http://javascript.crockford.com/). JSON - текстовый формат, полностью независимый от языка реализации, но он использует соглашения, знакомые программистам C-подобных языков, таких как C, C++, C#, Java, JavaScript, Perl, Python и многих других. Эти свойства делают JSON идеальным языком обмена данными.

Данные в формате JSON представляют собой:

* JavaScript-объекты { ... } или
* Массивы [ ... ] или
* Значения одного из типов:
  + строки в двойных кавычках,
  + число,
  + логическое значение true/false,
  + null.

Преимуществом данного метода обмена данными является то, что почти все языки программирования имеют библиотеки для преобразования объектов в формат JSON. Настройки пользователя будут подробнее рассмотрены на этапе физического проектирования базы данных.

Выше было выделено восемь задач. Задачи характеризуются типом.

Выделяют два типа задач:

1. Транзакции – задачи, изменяющие данные (добавление, удаление, обновление).
2. Запросы – задачи, совершающие выборку данных из базы, не влияя на ее состояние.

Первые четыре задачи - транзакции.

*Первая задача* имеет следующие входные данные: имя пользователя, пароль. На выходе получим нового основного пользователя в базе, или ошибку.

*Вторая задача* имеет следующие входные данные: имя пользователя, настройки в формате JSON. На выходе, у существующего пользователя появятся связанные с ним настройки сервиса.

*Третья задача* имеет следующие входные данные: имя пользователя, адрес устройства, длительность регистрации. Выходные данные: новое привязанное к основному пользователю устройство.

*Четвертая задача* имеет следующие входные данные: имя пользователя, адрес устройства. Выходные данные: связь устройства с основным пользователем удалена из базы.

Задачи с пятой по восьмую являются запросами.

*Пятая задача* это запрос, возвращающий настройки пользователя по его идентифицирующему полю.

*Шестая задача* это запрос, возвращающий информацию о том, существует ли данный пользователь в списке первичных пользователей в базе данных.

*Седьмая задача* это запрос, возвращающий информацию о том, есть ли в базе данных связанные с данным первичным пользователем устройства (зарегистрированные на него).

*Восьмая задача* это запрос, возвращающий выборку по имени пользователя, включающую все зарегистрированные на него устройства.

Описав все задачи, необходимо составить словарь данных. Словарь данных приведен в таблице 2.2.

Таблица 2.2 – Словарь данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Идентификатор | Описание назначения | Размер | Форма представления | Перечень задач, в которых используется |
| user\_name | Уникальное имя первичного пользователя в формате SIP URI | 255 | Символьная | 1 - 8 |
| settings | Настройки сервиса | 1024 | Символьная (JSON) | 2, 5 |
| binding | Адрес устройства, зарегистрированного на первичного пользователя | 255 | Символьная | 3, 4, 8 |
| expires | Время истечения срока регистрации, выраженное в количестве секунд с *00:00:00 UTC 1 января 1970 года* | 15 | Числовая | 3, 4, 8 |

Далее необходимо выделить набор сущностей.

Сущность – любой конкретный или абстрактный объект в рассматриваемой предметной области. Сущности – это базовые типы информации, которые хранятся в БД (в реляционной БД каждой сущности назначается таблица). К сущностям могут относиться: студенты, клиенты, подразделения и т.д. Каждая сущность обладает связями и атрибутами.

Атрибут – это свойство сущности в предметной области. Его наименование должно быть уникальным для конкретного типа сущности. Например, для сущности студент могут быть использованы следующие атрибуты: фамилия, имя, отчество, дата и место рождения, паспортные данные и т.д. В реляционной БД атрибуты хранятся в полях таблиц.

В таблице 2.3 приведен список сущностей с соответствующими им идентификаторами.

Таблица 2.3 – Сущности

|  |  |
| --- | --- |
| Сущность | Идентификатор сущности |
| Первичные пользователи | users |
| Настройки пользователя | settings |

Продолжение таблицы 2.3

|  |  |
| --- | --- |
| Сущность | Идентификатор сущности |
| Зарегистрированные устройства | bindings |

Сущность Первичные пользователи отображает существующих абонентов телефонной сети, и участвует во всех задачах.

Сущность Настройки пользователя отображает настройки сервиса, применимые к конкретному первичному пользователю, и участвует в задачах 2, 5.

Сущность Зарегистрированные устройства отображает адреса конкретных устройств, привязанных к определенному первичному пользователю путем регистрации по протоколу SIP, и участвует в задачах 3,4,7,8.

Определив сущности и подробно описав их назначения, необходимо составить описательные атрибуты для сущностей. Добавление описательных атрибутов приведено в таблице 2.4.

Таблица 2.4 – Сущности с описательными атрибутами

|  |  |  |
| --- | --- | --- |
| Сущность | Идентификатор сущности | Описательные атрибуты |
| 1 | 2 | 3 |
| Первичные пользователи | users | user\_name (имя пользователя), password (пароль) |
| Настройки пользователя | settings | user\_name, service\_settings (настройки пользователя в формате JSON) |
| Зарегистрированные устройства | bindings | user\_name, binding (адрес зарегистрированного устройства), expires (время истечения срока регистрации) |

Имея все необходимые сущности, их описательные атрибуты, можно составить концептуальную модель. Схема концептуальной модели изображена на рисунке 2.2.

bindings

users

settings

1:1 1:N

Рисунок 2.2 – Схема концептуальной модели БД

В результате концептуального проектирования выявлена зависимость типа *один к одному* между сущностями settings и users. Такую зависимость можно разрешить объединив две сущности в одну: users и settings в сущность users, и атрибут service\_settings сущности settings сделав атрибутом сущности users. Результирующий набор сущностей и их атрибутов отражен в таблице 2.5.

Таблица 2.5 – Уточненные сущности с описательными атрибутами

|  |  |  |
| --- | --- | --- |
| Сущность | Идентификатор сущности | Описательные атрибуты |
| 1 | 2 | 3 |
| Первичные пользователи | users | user\_name (имя пользователя), password (пароль), service\_settings (настройки пользователя в формате JSON) |
| Зарегистрированные устройства | bindings | user\_name, binding (адрес зарегистрированного устройства), expires (время истечения срока регистрации) |

Схема уточненной концептуальной модели изображена на рисунке 2.3.

users

bindings

1:N

Рисунок 2.3 – Схема уточненной концептуальной модели БД

**Логическое проектирование**

После успешного создания концептуальной модели можно к построению логической модели.

Логическое проектирование – преобразование требований к данным в структуры данных. На выходе получаем СУБД-ориентированную структуру базы данных и спецификации прикладных программ. На этом этапе часто моделируют базы данных применительно к различным СУБД и проводят сравнительный анализ моделей.

Логическая модель создается на основе уже построенной концептуальной модели. Создать логическую модель можно при помощи СУБД. Логическая модель изображена на рисунке 2.4.

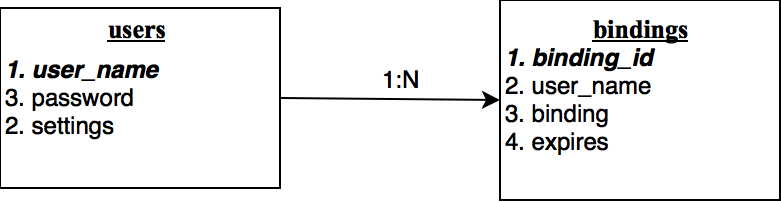


Рисунок 2.4 – Логическая модель БД

**Физическое проектирование**

Физическое проектирование представляет собой создание готовых для использования таблиц на основе построенной логической модели БД.

В таблице «Первичные пользователи» (users) содержится информация о пользователях (абонентах сети). Физическая структура таблицы «Первичные пользователи» приведена в таблице 2.5.

Таблица 2.6 – Физическая структура «Первичные пользователи»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название поля | Тип | Размерность | Первичный ключ | Вторичный ключ |
| user\_name | VARCHAR | 255 | да | - |
| settings | JSON | - | - | - |
| Printer\_name | VARCHAR | 20 | - | - |

В таблице «Зарегистрированные устройства» (bindings) содержатся адреса конкретных устройств, привязанных к определенному пользователю. Физическая структура таблицы «Зарегистрированные устройства» приведена в таблице 2.7.

Таблица 2.7 – Физическая структура «Зарегистрированные устройства»

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название поля | Тип | Размерность | Первичный ключ | Внешний ключ |
| binding\_id | INTEGER | - | да | - |

Продолжение таблицы 2.7

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название поля | Тип | Размерность | Первичный ключ | Внешний ключ |
| user\_name | VARCHAR | 255 | - | да  (users.user\_name) |
| binding | VARCHAR | 255 | - | - |
| expires | INTEGER | - | - | - |

*Структура настроек пользователя*

Настройки пользователя будут представлены в формате JSON. Для описания структуры настроек пользователя необходимо выделить сущности JSON объекта настроек.

Объект настроек будет включать следующий набор сущностей:

* service-name – строка, содержащая название сервиса, в данном случае «flexible-communication».
* service-settings – JSON объект, содержащий непосредственно настройки.

В таблице 2.8 приведена структура JSON-объекта service-settings.

Таблица 2.8 – структура JSON-объекта «service-settings»

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Назначение |
| not-reachable-timer | Число | Стандартный период времени в секундах, в течение которого от пользовательского устройства будет ожидаться первичный ответ (устройство пользователя сигнализирует о том что оно доступно в сети). |
| active | Логическое значение | Логическое значение, определяющее активен сервис для данного пользователя или нет. |
| default-period | Число | Стандартный период времени в секундах, в течение которого будет ожидаться положительный ответ от пользователя (пользователь снимет трубку). |
| target-list | JSON-массив | Содержит объекты target (цель), характеризующие устройства на которые может быть перенаправлен вызов. |

Продолжение таблицы 2.8

|  |  |  |
| --- | --- | --- |
| Название поля | Тип | Назначение |
| target | JSON-объект | Содержит адрес пользователя, и его короткое имя, на которое будут ссылаться другие сущности объекта настроек. |
| rule-set | JSON-массив | Содержит в себе массив объектов rule (правило). |
| rule | JSON-объект | Содержит в себе условия (conditions) выполнения данного правила, и набор действий (actions), которые будут выполнены в зависимости от того были соблюдены условия, или нет. |

**Описание объекта rule**

Основным элементом настроек является JSON-объект: rule.

Объект rule включает в себя массив conditions, содержащий набор условий (condition), и массив action-set, содержащий набор выполняемых действий (action).

***Действия (action)***

Действие в своей сути это вызов одного или нескольких устройств, описанных ранее в списке устройств (target-list), в параллельном или последовательном режиме.

Параллельный режим подразумевает вызов сразу нескольких целей и характеризуется общим периодом дозвона (period). Как только одна из вызываемых целей сняла трубку, вызов ко всем остальным завершается, путем отправки SIP сообщения CANCEL.

Последовательный режим подразумевает вызов целей одна за другой и так же характеризуется общим периодом дозвона (period), однако у каждой цели в последовательном режиме может быть указан собственный период дозвона, в противном случае используется общий. Если цель не приняла вызов в течение периода дозвона, то вызов данной цели будет завершен путем отправки SIP сообщения CANCEL и сервис приступит к вызову следующей цели (если таковые имеются). В противном случае.

Если ни одна из целей параллельного или последовательного действия не приняла вызов, то сервис приступает к выполнению следующего действия данного правила, которое может быть как последовательным, так и параллельным.

Сценарий с одним последовательным или параллельным действием в рамках одного правила называется сценарием *последовательного* или *параллельного перенаправления вызовов* соответственно.

Сценарий с несколькими действиями в рамках одного правила называется сценарием *гибкого перенаправления вызовов*.

***Условия (Conditions)***

Для того чтобы сервис приступил к выполнению действий (action) указанных в правиле (rule), необходимо чтобы все условия (condition), указанные в данном правиле, были удовлетворены.

Условия представлены в виде массива JSON объектов. Каждое условие характеризуется строкой condition, обозначающей идентификатор условия. Так же один из видов условий помимо идентификатора характеризуется значением, этот тип обозначает периоды валидности данного правила, и называется valid-periods. Подробнее данный тип и его значения будут описаны ниже.

Существуют четыре типа условий:

1. not-reachable – не доступен.
2. Busy – занят.
3. no-answer – не отвечает.
4. valid-periods – валидные периоды.

*not-reachable*

Данное условие выполняется в том случае, когда пользователь зарегистрирован, но его устройство не доступно в сети. В такой ситуации после отправки запроса устройству, оно не вернет никакого промежуточного ответа, обозначающего, что запрос принят и поступил на обработку. Для определения выполняется ли данное условие, в приложении будет присутствовать таймер (not-reachable-timer), который был описан в таблице 2.8. Этот таймер будет установлен в момент отправки сообщения INVITE, и если таймер закончится раньше чем придет хоть какой то промежуточный ответ от вызываемого устройства, то условие считается удовлетворенным. Если же промежуточный ответ придет раньше, то таймер будет остановлен.

*busy*

Данное условие выполняется в том случае, если устройство пользователя шлет ответ 486 Busy. Это может означает что пользователь сбросил вызов, или же он занят разговором с другим пользователем.

*no-answer*

Данное условие выполняется в том случае, если вызываемый пользователь уже прислал промежуточный ответ 180 Ringing (устройство проигрывает мелодию вызова) но не прислал конечный ответ в течение определенного промежутка времени. Промежуток времени выбирается на основании периода, указанного в действии, или же, если это первичный запрос, то на основании значения поля default-period, описанного выше в таблице 2.8

*valid-periods*

Данное условие задает временные ограничения в виде дня (дней) недели, и периода времени дня в 24 часовом формате.

Условие valid-periods должно содержать (помимо идентификатора условия) JSON-массив periods.

Массив periods содержит в себе набор периодов, характеризующихся типом периода и набором значений.

Есть два типа периодов:

1. valid-days – валидные дни. Содержит список дней когда данное условие валидно.
2. valid-times – валидные времена суток. Выраженные в 24 часовом формате временные промежутки в которые данное условие валидно. Если текущее время попадает хоть в один из промежутков, то условие выполнено.

Ниже приведен пример настроек пользователя.

{

"service-name" : "flexible-communication",

"service-settings" : {

"not-reachable-timer" : "4",

"active" : "true",

"default-period" : "7",

"target-list": [

{

"name" : "Dave",

"id" : "sip:Dave@127.0.0.1:5061"

},

{

"name" : "Carol",

"id" : "sip:Carol@127.0.0.1:5070"

},

{

"name" : "Elvis",

"id" : "sip:Elvis@127.0.0.1:5071"

},

{

    "name" : "Fin",

"id" : "sip:Fin@127.0.0.1:5062"

}

],

"rule-set" : [

{

"id" : "test",

"conditions" : [

{

"condition" : "not-reachable"

},

{

"condition" : "valid-periods",

"periods" : [

{

"period-type" : "valid-days",

"values" : [

"Monday",

"Tuesday",

"Wednesday",

"Thursday",

"Friday"

]

},

{

"period-type" : "valid-times",

"values" : [

{

"from" : "00:00",

"to" : "24:00"

}

]

}

]

}

],

"action-set" : [

    {

"action-type" : "serial-ringing",

"period" : "5",

"targets" : [

{

"target" : "Elvis",

"period" : 3

},

{

"target" : "Dave",

"period" : 5

}

]

},

{

"action-type" : "parallel-ringing",

"period" : 6,

"targets" : [

{

"target" : "Fin"

},

{

"target" : "Carol"

}

]

}

]

}

]

}

}

Физическое проектирование базы данных было последним этапом проектирования, так как именно данный этап отражает готовую структуру базы данных.

Объединение всех вышеперечисленных этапов и является построением структуры информационного обеспечения.

## 2.4 Структура пользовательского интерфейса

Пользовательский интерфейс объединяет в себе все элементы и компоненты программы, с помощью которых пользователь может взаимодействовать с программным обеспечением.

К этим элементам относятся:

* средства отображения информации, отображаемая информация, форматы и коды;
* командные режимы, язык пользователь-интерфейс;
* устройства и технологии ввода данных;
* диалоги, взаимодействие и транзакции между пользователем и компьютером;
* обратная связь с пользователем;
* поддержка принятия решений в конкретной предметной области;
* порядок использования программы и документация на нее.

Интерфейс пользователя — система правил и средств, регламентирующая и обеспечивающая взаимодействие программы с пользователем.

В рамках данного дипломного проекта не предусматривается разработка клиентского программного обеспечения.

Для взаимодействия пользователя с реализуемой системой, ему понадобится клиентское устройство – так называемый sip-софтфон (sip-softphone).

На рынке доступно большое количество sip-софтфонов, как платных, так и бесплатных.

Из бесплатных решений можно выделить например такие как:

* MicroSIP – очень простой легковесный клиент, с открытым исходным кодом на языке C и C++. Доступен только для ОС Windows.
* Phoner lite – бесплатный клиент для Windows с закрытым исходным кодом. Отличается от MicroSIP более продвинутым и удобным графическим интерфейсом.
* QuteCom – это мощный sip-клиент, для операционных систем Linux.

На рисунках 2.5 и 2.6 представлены примеры sip-клиентов.

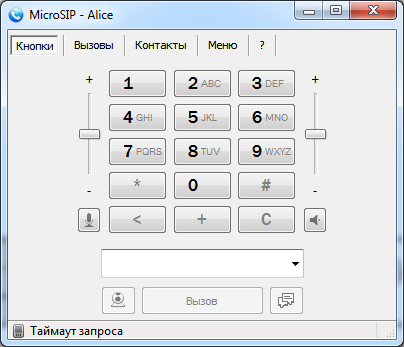


Рисунок 2.5 – Интерфейс MicroSIP

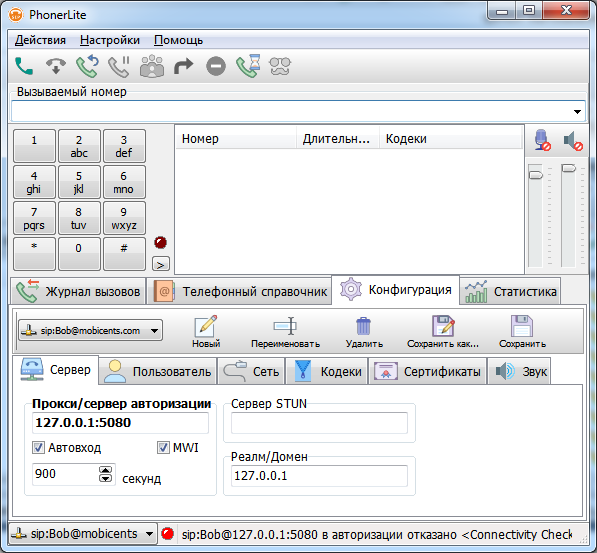


Рисунок 2.6 – Интерфейс PhonerLite

# 3 РЕАЛИЗАЦИЯ И ИСПЫТАНИЕ СИСТЕМЫ

## 3.1 Выбор средств реализации

В качестве средств разработки программного обеспечения было решено выбрать следующий набор инструментов:

* Язык программирования Java;
* Фрэймворк RestComm Sip Servlets;
* PostgreSQL.

Рассмотрим перечисленные выше технологии более подробно.

**Язык программирования Java**

На сегодняшний момент язык Java является одним из самых распространенных и популярных языков программирования. Первая версия языка появилась еще в 1996 году в недрах компании Sun Microsystems, впоследствии поглощенной компанией Oracle. Java задумывался как универсальный язык программирования, который можно применять для различного рода задач. И к настоящему времени язык Java проделал большой путь, было издано множество различных версий. Текущей версией является Java 8, официальный релиз которой произошел в марте 2014 года. А Java превратился из просто универсального языка в целую платформу и экосистему, которая объединяет различные технологии, используемые для целого ряда задач: от создания настльных приложений до написания крупных веб-порталов и сервисов. Кроме того, язык Java активно применяется для создания программного обеспечения для целого ряда устройств: обычных ПК, планшетов, смартфонов и мобильных телефонов и даже бытовой техники. Достаточно вспомнить популярность мобильной ОС Android, большинство программ для которой пишутся именно на Java.

*Особенности Java*

Ключевой особенностью языка Java является то, что его код сначала транслируется в специальный байт-код, независимый от платформы. А затем этот байт-код выполняется виртуальной машиной JVM (Java Virtual Machine). В этом плане Java отличается от стандартных интерпретируемых языков как PHP или Perl, код которых сразу же выполняется интерпретатором. В то же время Java не является и чисто компилируемым языком, как С или С++.

Подобная архитектура обеспечивает кроссплатформенность и аппаратную переносимость программ на Java, благодаря чему подобные программы без перекомпиляции могут выполняться на различных платформах - Windows, Linux, Solaris и т.д. Для каждой из платформ может быть своя реализация виртуальной машины JVM, но каждая из них может выполнять один и тот же код.

Java является языком с Си-подобным синтаксисом и близок в этом отношении к C/C++ и C#.

Еще одной ключевой особенностью Java является то, что он поддерживает автоматическую сборку мусора. А это значит, что разработчику не надо освобождать вручную память от ранее использовавшихся объектов, как в С++, так как сборщик мусора это сделает автоматически.

Java является объектно-ориентированным языком. Он поддерживает полиморфизм, наследование, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений.

**RestComm SIP Servlets**

RestComm SIP Servlets это SIP, IMS сервер приложений с открытым исходным кодом. Restcomm SIP Servlets - это современная коммуникационная платформа промежуточного уровня. RestcommSIP Servlets облегчает переход к облачным коммуникациям, позволяя развертывать и автоматически настраивать SIP сервлет приложения на базе всех основных IaaS (Infrastructure as a Service) провайдеров, а также обеспечивает связь в реальном времени (голос и видео) через браузер с использованием HTML5 WebRTC и SIP Over WebSockets.

Сервер Restcomm SIP Servlets реализуют стандарт SIP Servlet v1.1 (JSR 289). Он может быть подключен к любому контейнеру сервлетов (в настоящее время Apache Tomcat 8.X и JBoss 7.X), а также обеспечивает высокую доступность и отказоустойчивость.

В рамках данного проекта будет использован контейнер Apache Tomcat 8.0.26 с подключенным к нему фрэймворком Restcomm SIP Servlets версии 4.0.119.

**PostgreSQL**

PostgreSQL — это объектно-реляционная система управления базами данных (ОРСУБД, ORDBMS), основанная на [POSTGRES, Version 4.2](http://db.cs.berkeley.edu/postgres.html) — программе, разработанной на факультете компьютерных наук Калифорнийского университета в Беркли. В POSTGRES появилось множество новшеств, которые были реализованы в некоторых коммерческих СУБД гораздо позднее.

PostgreSQL — СУБД с открытым исходным кодом, основой которого был код, написанный в Беркли. Она поддерживает большую часть стандарта SQL и предлагает множество современных функций:

* Сложные запросы;
* Внешние ключи;
* Триггеры;
* Изменяемые представления;
* Транзакционная целостность;
* Многоверсионность.

Кроме того, пользователи могут всячески расширять PostgreSQL, например, создавая свои

* Типы данных;
* Функции;
* Операторы;
* Агрегатные функции;
* Методы индексирования;
* Процедурные языки.

А благодаря свободной лицензии, PostgreSQL разрешается бесплатно использовать, изменять и распространять всем и для любых целей — личных, коммерческих или учебных.

## 3.2 Реализация системы

Реализация системы включает в себя реализацию программного и информационного обеспечения системы.

Разработка информационного обеспечения была в полной мере описана в разделе 2.3. В вышеупомянутом разделе была спроектирована физическая модель базы данных, описаны создаваемые таблицы и их атрибуты. Так же была подробно описана структура пользовательских настроек, хранимых в базе данных в формате JSON.

Структура ПО была разработана и описана в подразделе 2.2.

* Была описана структура модулей разрабатываемого ПО (таблица 2.1).
* Была описана модель программирования, применяемая при разработке приложений на основе спецификации Sip Servlet.
* Был описан паттерн State (состояние).

Однако для полноценного описания реализации системы необходимо описать реализуемую с помощью паттерна State машину состояний, которая является основным элементом логики приложения.

В ходе разработки приложения было выделено девять состояний:

1. INITIAL\_STATE – начальное состояние. Когда сервис получает первое сообщение, машина состояние находится именно в INITIAL\_STATE.
2. NO\_SETTINGS – это состояние устанавливается после INITIAL\_STATE, если вызываемый пользователь зарегистрирован, но не подписан на услугу *гибкого перенаправления вызовов*.
3. NO\_RULES\_MATCHED – это состояние означает, что после обработки инициирующего диалог сообщения было определено что вызываемый пользователь подписан на услугу *гибкого перенаправления вызовов,* однако ни одно из правил не было применено и сообщение было просто перенаправлено на зарегистрированное устройство вызываемого абонента.
4. SERIAL\_RINGING – это состояние означает, что для вызываемого пользователя был задействован сервис *гибкого перенаправления вызовов,* и активировано действие *последовательного перенаправления*. В данном состоянии сервис может оказаться сразу после INITIAL\_STATE, после NO\_RULES\_MATCHED, или же, в случае сценария *гибкого перенаправления вызовов,* после состояния PARALLEL\_RINGING, при активировании следующего действия.
5. PARALLEL\_RINGING – это состояние означает, что для вызываемого пользователя был задействован сервис *гибкого перенаправления вызовов,* и активировано действие *параллельного перенаправления*. В данном состоянии сервис может оказаться сразу после INITIAL\_STATE, после NO\_RULES\_MATCHED, или же, в случае сценария *гибкого перенаправления вызовов,* после состояния SERIAL\_RINGING, при активировании следующего действия.
6. INVITE\_ACCEPTED – это состояние означает, что одно из вызываемых устройств приняло вызов вернув положительный ответ 200 Ok, и этот положительный ответ был перенаправлен вызывающему пользователю. В данном состоянии сервис может оказаться после NO\_SETTINGS\_STATE, после NO\_RULES\_MATCHED, после PARALLEL\_RINGING или же после SERIAL\_RINGING.
7. DIALOG\_INSTALLED – это состояние означает, что вызывающий пользователь получил положительный ответ, перенаправленный ему от вызываемого пользователя, и подтвердил установление диалога сообщением ACK (в соответствии со спецификацией протокола SIP). В этом состоянии сервис может оказаться только после состояния INVITE\_ACCEPTED.
8. INVITE\_CANCELLED – это промежуточное состояние, которое означает что вызов по каким то причинам прерван до установления диалога, либо вызывающим абонентом, либо вызываемым, либо же ошибкой самого сервиса или сети. Это состояние служит для освобождения ресурсов и для корректного завершения сеанса связи. Оно является конечным.
9. DIALOG\_TERMINATED – это состояние означает что состоявшийся диалог был завершен, например когда один из участников диалога отправил запрос BYE, а второй ответил 200 Ok (в соответствии со спецификацией протокола SIP). Оно является конечным.

Описанная выше машина состояний служит для реализации основного функционала сервиса – *гибкого перенаправления вызовов*,или в случае если пользователь не подписан на данный сервис, для установки обычного диалога. Функционал регистрации не использует шаблон State (состояние) и реализован с помощью класса обертки над необходимыми для регистрации запросами к базе данных. Экземпляр данного класса используется в методе doRegister главного сервлета приложения.

Результаты испытаний системы приведены в разделе 3.3.

## 3.3 Результаты испытаний

Целью данного раздела является проверка работоспособности функций системы, описанных в постановке задачи. А так же проверка характеристик системы.

Тестирование разделено на три раздела:

1. Тестирование базового функционала (без настроек сервиса):
   1. Регистрация / Де-регистрация;
   2. Звонок первичному пользователю, без зарегистрированных устройств;
   3. Звонок пользователю, устройство которого занято;
   4. Завершение звонка вызывающим абонентом, до установления диалога;
   5. Успешное установление диалога;
   6. Завершение звонка по истечению таймера вызова.
2. Тестирование функционала перенаправления вызовов:
   1. Последовательное перенаправление вызовов;
   2. Параллельное перенаправление вызовов;
3. Тестирование функционала гибкого перенаправления вызовов и проверки условий выполнения правил:
   1. Параллельное-последовательное перенаправление при условии on-busy;
   2. Последовательное-параллельное перенаправление при условии no-answer;

Тестирование проводилось на устройстве со следующими характеристиками:

* процессор – Intel Core i7-4500U CPU @1.80GHz;
* ОЗУ – 8ГБ;
* тип системы – 64 разрядная;
* монитор – 15 дюймов;
* HDD – 500GB;
* клавиатура, мышь;

На данную машину был установлен контейнер Apache Tomcat 8.0.26 с подключенным к нему фрэймворком Restcomm SIP Servlets версии 4.0.119.

Разработанное приложение было установлено в данный контейнер и сервер был запущен и настроен на прием SIP запросов по протоколам (UDP/TCP) на порту 5080.

Для тестирования локально были установлены три SIP-софтфона и настроены на адрес SIP-сервера 127.0.0.1:5080.

Для демонстрации результатов тестирования была использована программа Wireshark.

Wireshark — [программа-анализатор трафика](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%D0%B0%D1%82%D0%BE%D1%80_%D1%82%D1%80%D0%B0%D1%84%D0%B8%D0%BA%D0%B0) для компьютерных сетей [Ethernet](https://ru.wikipedia.org/wiki/Ethernet" \o "Ethernet) и некоторых других.

Функциональность, которую предоставляет Wireshark, очень схожа с возможностями программы [tcpdump](https://ru.wikipedia.org/wiki/Tcpdump" \o "Tcpdump), однако Wireshark имеет [графический пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81) и гораздо больше возможностей по сортировке и фильтрации информации. Программа позволяет пользователю просматривать весь проходящий по сети трафик в режиме реального времени, переводя сетевую карту в [неразборчивый режим](https://ru.wikipedia.org/wiki/Promiscuous_mode) ([promiscuous mode](https://en.wikipedia.org/wiki/Promiscuous_mode" \o "en:Promiscuous mode)).

Дополнительно был установлен набор библиотек Npcap, которые позволяют захватывать так называемый loopback трафик. Loopback — это термин, который обычно используется для описания методов или [процедур](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0) [маршрутизации](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) электронных сигналов, цифровых [потоков данных](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D0%BE%D0%BA_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85), или других движущихся сущностей от их источника и обратно к тому же источнику без специальной обработки или модификаций. Первоначально он использовался для тестирования передачи или передающей инфраструктуры.

Ниже приведены результаты тестирования в виде Wireshark логов и их описания.

**Тестирование базового функционала (без настроек сервиса)**

Все описанные в этом разделе тестовые сценарии подразумевают, что вызываемый пользователь не подключил услугу гибкого перенаправления вызовов.

***Регистрация / Де-регистрация***

Процесс регистрации и де-регистрации продемонстрирован на рисунках 3.1 и 3.2

Устройство посылает запрос REGISTER и получает ответ 200 Ok, что свидетельствует об успешной регистрации.

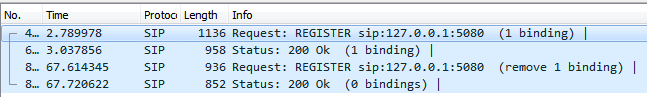


Рисунок 3.1 – Регистрация и Де-регистрация

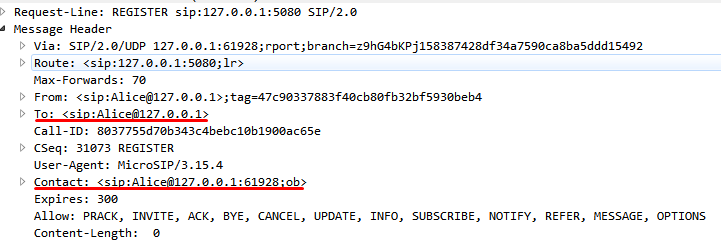


Рисунок 3.2 – Поля сообщения регистрации

Обратите внимание на выделенные красным цветом поля на рисунке 3.2. В запросе на регистрацию устройство по адресу 127.0.0.1:61928 (поле contact) регистрируется на первичного пользователя sip:Alice@127.0.0.1 (поле To).

Результатом регистрации является добавление записи в таблице bindings (связи). Результат в виде запроса всех данных из таблицы bindings представлен на рисунке 3.3

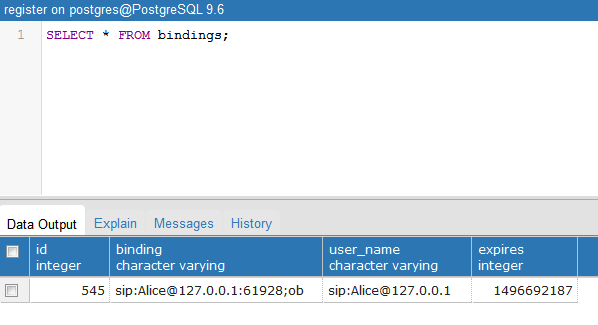


Рисунок 3.3 – результат регистрации.

Как видно из рисунка 3.3 в таблице появилась связь устройства по адресу 127.0.0.1:61928 с первичным пользователем sip:Alice@127.0.0.1; Следовательно регистрация прошла успешно.

Далее за регистрацией следует де-регистрация, в результате которой из базы удаляется запись связи адреса устройства с первичным пользователем. Следовательно де-регистрация прошла успешно.

***Звонок первичному пользователю без зарегистрированных устройств***

Сценарий звонка продемонстрирован на рисунке 3.4

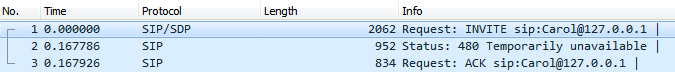


Рисунок 3.4 – Вызов незарегистрированного пользователя

Как видно из рисунка 3.4, сервис получил запрос INVITE на вызов пользователя sip:Carol@127.0.0.1, который не был ранее зарегистрирован, и в результате вызывающее устройство получило ответ 480 Временно недоступен, свидетельствующий о том, что у вызываемого пользователя нет ни одного зарегистрированного устройства.

***Звонок пользователю, устройство которого занято***

При вызове устройства, оно может вернуть ответ 486 Busy Here (Занято), в таком случае, учитывая что у вызываемого пользователя не подключена функция *гибкого перенаправления вызовов*, необходимо перенаправить этот ответ вызываемому пользователю. Данный сценарий продемонстрирован на рисунке 3.5

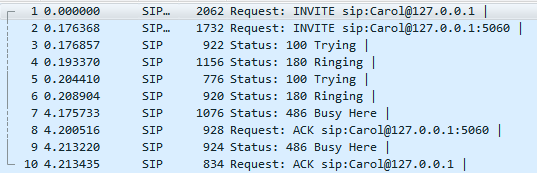


Рисунок 3.5 – Вызываемое устройство занято

Из рисунка 3.5 видно, что вызов первичного пользователя sip:Carol@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip:Carol@127.0.0.1:5060; Затем через некоторое время вызываемое устройство возвращает ответ 486 Busy Here на который сразу же получает подтверждение от сервера (ACK). После чего данный ответ перенаправляется на вызывающее устройство, которое так же высылает подтверждение (ACK) и сессия завершается.

***Завершение звонка вызывающим абонентом, до установления диалога***

Вызывающий абонент может прервать вызов до получения ответа об успешном установлении диалога. Это делается путем отправки сообщения CANCEL. Данный сценарий продемонстрирован на рисунке 3.6

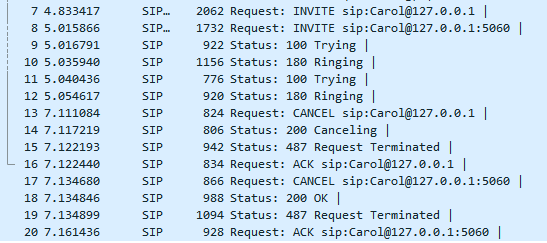


Рисунок 3.6 – Завершение раннего диалога

Из рисунка 3.6 видно, что вызов первичного пользователя sip:Carol@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip:Carol@127.0.0.1:5060; Затем через некоторое время вызывающий абонент нажимает на кнопку сброса (сообщение CANCEL). Сервер получив данное сообщение возвращает 200 Canceling и сразу после этого 487 Request Terminated, все в соответствии со стандартом SIP. После чего сообщение CANCEL перенаправляется вызываемому абоненту и сессия завершается таким же образом как и описано ранее.

***Успешное установление диалога***

Успешное установление диалога заключается в отправке вызываемым устройством ответа 200 Ok на запрос INVITE, и после подтверждение вызывающим устройством принятия данного ответа, с помощью сообщения ACK. Данный сценарий продемонстрирован на рисунке 3.7

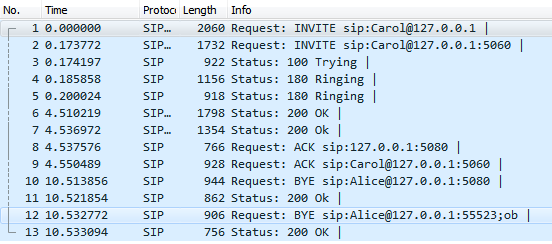


Рисунок 3.7 – SIP диалог

Из рисунка 3.7 видно, что вызов первичного пользователя sip:Carol@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip:Carol@127.0.0.1:5060; Затем через некоторое время вызываемый абонент отправляет ответ 200 Ok, который перенаправляется вызывающему пользователю. Вызывающее устройство подтверждает установку диалога с помощью сообщения ACK. Диалог установлен. Через некоторое время один из участников диалога «кладет трубку». Отправляется запрос BYE и диалог завершается.

***Завершение звонка по истечению таймера вызова.***

Когда устройство получает запрос на установление диалога оно обычно начинает проигрывать сигнал вызова, и чтобы вызывающее устройство об этом узнало, вызываемое устройство отправляет ответ 180 Ringing. При получении этого ответа вызываемое устройство начинает проигрывать пользователю длинные гудки.

Вызываемый пользователь может не слышать сигнал вызова и быть не способным снять трубку, поэтому необходимо ограничить период вызова чтобы он не продолжался слишком долго.

Для тестирования данной функциональности стандартный таймер завершения вызова был установлен в 10 секунд. Сценарий завершения вызова по таймеру продемонстрирован на рисунке 3.8

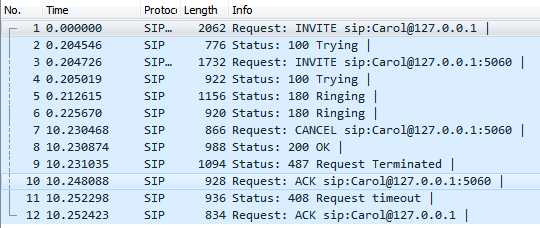


Рисунок 3.8 – Завершение вызова по истечении таймера

Из рисунка 3.8 видно, что вызов первичного пользователя sip:Carol@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip:Carol@127.0.0.1:5060; Затем по истечении десяти секунд, как указано в настройках таймера, сервер шлет запрос завершения раннего диалога (сообщение CANCEL) вызываемому устройству, которое отвечает на данный запрос в соответствии со стандартом. И затем сервер отправляет вызываемому устройству ответ 408 Request timeout, свидетельствующий о истечении

**Тестирование функционала перенаправления вызовов**

Вызываемый пользователь – sip:Bob@127.0.0.1 подключил себе сервис *гибкое перенаправление вызовов.*

Пользователь указал в настройках следующий список целей:

* Dave - <sip:Dave@127.0.0.1:5061>
* Carol - sip:Carol@127.0.0.1:5070
* Elvis - sip:Elvis@127.0.0.1:5071
* Fin - sip:Fin@127.0.0.1:5062

***Последовательное перенаправление вызовов***

Вызываемый пользователь настроил последовательное перенаправление вызовов на событие on-busy (занято, устройство возвращает ответ 487 Busy ).

В действии указано последовательное перенаправление вызовов на цели:

1. Elvis
2. Dave
3. Carol

Данный сценарий продемонстрирован на рисунке 3.9

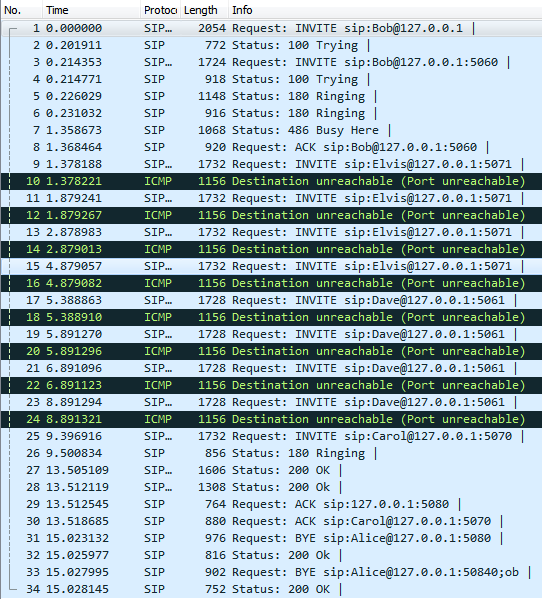


Рисунок 3.9 – Последовательное перенаправление вызовов

Из рисунка 3.9 видно, что вызов первичного пользователя sip:Bob@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip: Bob @127.0.0.1:5060; Затем Bob отправляет ответ 487 Busy, что заставляет выполниться функционалу *последовательного перенаправления вызовов.* Из рисунка 3.9 видно что пользователи Elvis и Dave вызываются последовательно (пакеты 9 - 24), однако так как устройства не включены, то пользователи не доступны. Затем вызов перенаправляется на последнего пользователя в списке – Carol, который запущен на localhost. Carol принимает вызов и диалог развивается по сценарию *успешное установление диалога,* описанному выше*.*

***Параллельное перенаправление вызовов***

Вызываемый пользователь настроил параллельное перенаправление вызовов на событие on-busy (занято, устройство возвращает ответ 487 Busy ).

В действии указано последовательное перенаправление вызовов на цели:

1. Elvis
2. Carol

Данный сценарий продемонстрирован на рисунке 3.10

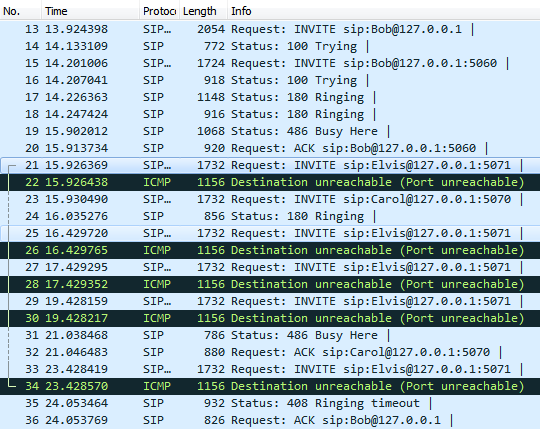


Рисунок 3.10 – Параллельное перенаправление вызовов

Из рисунка 3.10 видно, что вызов первичного пользователя sip:Bob@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip: Bob @127.0.0.1:5060; Затем Bob отправляет ответ 487 Busy, что заставляет выполниться функционалу *параллельного перенаправления вызовов.* Из рисунка 3.10 видно что пользователи Elvis и Carol вызываются параллельно (пакеты 21 - 34), однако так как устройство Elvis не включено, то пользователи не доступен. На пакете номер 31 Carol сбрасывает входящий вызов, но вызывающий пользователь не получает этот ответ (486 Busy), так как еще не закончен параллельный вызов устройства Elvis, Которое еще может ответить утвердительно на запрос установки диалога. Однако так как устройство Elvis недоступно, то по истечении таймера недоступности вызов завершается и вызывающему устройству возвращается ответ 408 Timeout.

**Тестирование функционала гибкого перенаправления вызовов и проверки условий выполнения правил**

Вызываемый пользователь – sip:Bob@127.0.0.1 подключил себе сервис *гибкое перенаправление вызовов.*

Пользователь указал в настройках следующий список целей:

* Dave - <sip:Dave@127.0.0.1:5061>
* Carol - sip:Carol@127.0.0.1:5070
* Elvis - sip:Elvis@127.0.0.1:5071
* Fin - sip:Fin@127.0.0.1:5062

***Параллельное-последовательное перенаправление при условии*** ***on-busy***

Вызываемый пользователь настроил гибкое перенаправление вызовов на событие on-busy (занято, устройство возвращает ответ 487 Busy ).

В действии указано параллельное перенаправление вызовов на цели:

1. Elvis
2. Dave

И затем последовательное перенаправление на цели:

* 1. Carol

Данный сценарий продемонстрирован на рисунке 3.11

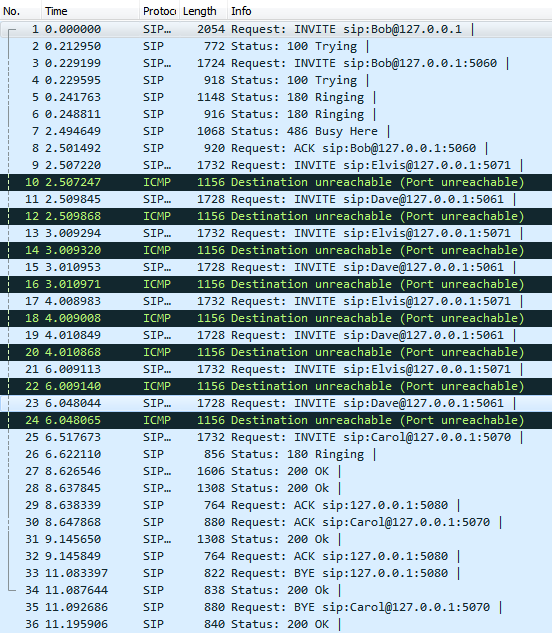


Рисунок 3.11 – Параллельное-последовательное перенаправление

Из рисунка 3.11 видно, что вызов первичного пользователя sip:Bob@127.0.0.1 попадая на обработку на сервер перенаправляется на зарегистрированное устройство данного пользователя по адресу sip: Bob @127.0.0.1:5060; Затем Bob отправляет ответ 487 Busy, что заставляет выполниться функционалу *параллельного перенаправления вызовов.* Из рисунка 3.11 видно что пользователи Elvis и Dave вызываются параллельно (пакеты 9 - 24), однако так как устройства не включены, то пользователи не доступны. На пакете номер 25 параллельный вызов прекращен и начат последовательный вызов устройства Carol. Carol принимает вызов и диалог развивается по сценарию *успешное установление диалога,* описанному выше*.*

***Последовательное-параллельное перенаправление при условии no-answer***

Вызываемый пользователь настроил гибкое перенаправление вызовов на событие on-no-answer (cрабатывает таймер вызова).

В действии указано последовательное перенаправление вызовов на цели:

1. Elvis
2. Dave

И затем параллельное перенаправление на цели:

1. Fin
2. Carol

Данный сценарий продемонстрирован на рисунке 3.12

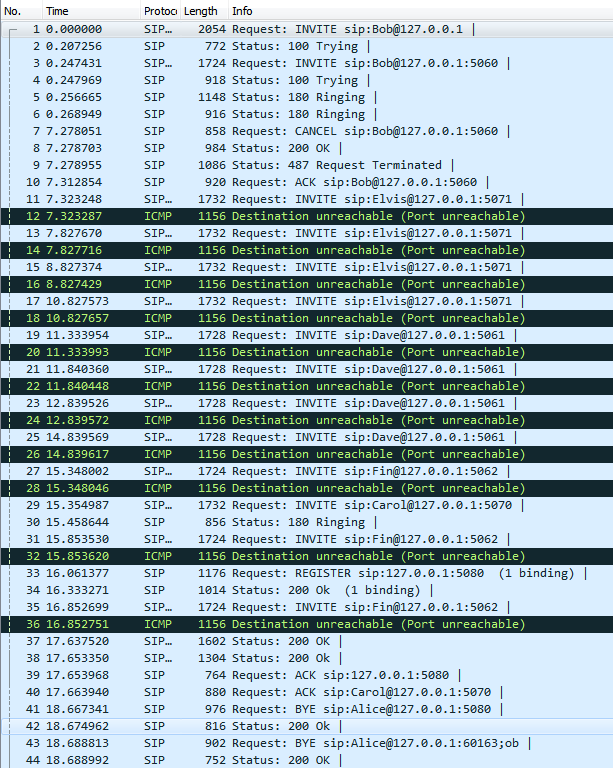


Рисунок 3.12 – Последовательно-параллельное перенаправление

Из рисунка 3.12 видно, что вызов первичного пользователя sip:Bob@127.0.0.1 сначала проходит по сценарию *Завершение звонка по истечению таймера вызова* (пакеты 1 - 10), однако вместо завершения вызова, выполняется последовательное перенаправление сначала на Elvis, затем на Dave (пакеты 11 – 26). Однако оба этих устройства не доступны, и по истечении таймера недоступности, вызов завершается и выполняется следующее действие – параллельный вызов Fin и Carol. Carol принимает вызов и диалог развивается по сценарию *успешное установление диалога,* описанному выше.

В результате система прошла все предусмотренные функциональные тесты. Это подтверждает полное соответствие возможностей созданной системы постановке задачи на дипломное проектирование.

# 4 РАСЧЕТ ЭКОНОМИЧЕСКОЙ ЭФФЕКТИВНОСТИ

## 4.1 Характеристика программного обеспечения

Наименование проекта – SIP сервер приложений с поддержкой функции «Гибкое перенаправление вызовов».

Среда разработки ПО – Java. Программное обеспечение функционального назначения. Система управления реляционными базами данных – PostgreSQL 9.6.

Общий объем ПО (Vо) определяется исходя из количества и объема функций, реализуемых программой:

|  |  |  |
| --- | --- | --- |
|  | | (4.1) |
| где | V0 – общий объем ПС;  Vi– объем функций ПС;  n – общее число функций. | |

Расчет общего объема ПО (количества строк исходного кода (LOC)) предполагает определение объема по каждой функции. В том случае, когда на стадии технико-экономического обоснования проекта невозможно рассчитать точный объем функций, то данный объем может быть получен на основании ориентировочной (прогнозной) оценки имеющихся фактических данных по аналогичным проектам, выполненным ранее, или путем применения нормативов по каталогу функций.

Определим уточненный объем ПО. На основании информации о функциях разрабатываемого ПО по каталогу функций определяется общий объем ПО. В зависимости от организационных и технологических условий, в которых разрабатывается ПО, был скорректирован объем на основе экспертных оценок.

Уточненный объем ПО (Vу) определяется по формуле:

|  |  |  |
| --- | --- | --- |
| Vу=∑ni=1Vуi, | | (4.2) |
| где | Vуi – уточненный объем отдельной функции в строках исходного кода (LOC) . | |

В таблице 4.1 приведен перечень и объем функций ПО, используемых в разработке.

Таблица 4.1 – Перечень и объем функций программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Код функции | Наименование (содержание) функции | Объем функции строк исходного кода (LOC) | |
| По каталогу Vi | Уточненный Vуi |
| 1 | 2 | 3 | 4 |
| 101 | Организация ввода информации | 130 | 86 |
| 202 | Генерация структуры базы данных | 3500 | 1203 |
|  | Формирование баз данных | 1980 | 560 |
| 207 | Организация поиска и поиск в базе данных | 4720 | 895 |
| 506 | Обработка ошибочных и сбойных ситуаций | 1540 | 356 |
| 507 | Обеспечение интерфейса между компонентами | 1680 | 776 |
| Итого: | | 13550 | 3876 |

Учитывая информацию, указанную в таблице 4.1, о функциях разрабатываемого программного обеспечения, уточненный объем ПО (Vу) составил 3876 строк исходного кода (LOC) вместо предполагаемы 13550 строк.

## 4.2 Расчет полной себестоимости программного продукта

Основная заработная плата определяется на основании разряда, тарифной ставки и отработанного времени

Основная заработная плата каждого исполнителя определяется по формуле:

ЗПо = Тст1 р \*Тк/ 22 \* Фрв \* Кпр, (4.3)

|  |  |
| --- | --- |
| где | Тст1 р – месячная тарифная ставка 1 разряда рабочего (с 1 января 2017 года на предприятии – 31 белорусский рубль);  Тк–тарифный коэффициент согласно разряду исполнителя;  22 – среднее количество рабочих дней в месяце;  Фрв – фонд рабочего времени исполнителя (продолжительность разработки ПП, дни);  Кпр – коэффициент премий. |

Определение часовой тарифной ставки производится по следующей формуле:

, (4.4)

где Тм.1р. – месячная тарифная ставка *i*-го исполнителя 1 разряда, BYN;

Прв – расчетная норма рабочего времени за год, час.

Дополнительная заработная плата включает в себя выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью работников).

Дополнительная заработная исполнителя рассчитывается от основной заработной платы по формуле:

(4.5)

где Нд - норматив дополнительной заработной платы, который можно принять в размере 10 – 20 %.

Аналогично основной, совокупная дополнительная заработная плата всех работников-создателей ПО/ПП (устройства) рассчитывается как сумма дополнительной зарплаты всех работников-участников проекта.

По данным на 01.05.2017 год месячная тарифная ставка первого разряда составляет 31 BYN. Среднее количество рабочих дней при пятидневной рабочей недели составляет 22 дня. Дополнительная заработная плата – 20% от основной заработной платы.

Результаты расчёта основной заработной платы исполнителя представлены в таблице 4.2.

Таблица 4.2 – Расчёт основной заработной платы исполнителей

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № | Категория исполнителя | Разряд | Тарифный коэффициент | Коэффициент премирования | Фонд рабочего времени, дн. | Нормативы дополнительной зарплаты, % | Заработная плата, BYN | | |
| Основная | Дополнительная | Всего |
| 1. | Программист 1-й категории | 10 | 2,48 | 1.6 | 35 | 16 | 195,7 | 31,3 | 227,0 |

2. Отчисления на социальные нужды (Рсоц) определяются в соответствии с действующим законодательством по нормативу (34% - отчисления в ФСЗН + 0,6% отчисления по обязательному страхованию):

 (4.6)

3. Расходы по статье «Спецоборудование» (Рс) включает затраты на приобретение технических и программных средств специального назначения, необходимых для разработки конкретного ПП, включая расходы на проектирование, изготовление, отладку и др. В данном дипломном проекте для разработки ПП приобретение какого-либо спецоборудования не предусматривалось. Так как спецоборудование не было приобретено, данная статья не рассчитывается.

4. По статье «Материалы и комплектующие изделия» (Рмк) отражаются расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки ПП. Норма расхода материалов в суммарном выражении определяются либо в расчете на 100 строк исходного кода (объём строк ПП определяется по Приложению 2), либо в процентах к основной заработной плате разработчиков, как правило, 3-5%.

Сумма затрат на расходные материалы рассчитывается по формуле:

Рмк = Нм \* (Vo /100), (4.7)

где Нм – норма расхода материалов в расчете на 100 строк исходного кода;

Vo – уточнённый общий объём функций строк исходного кода (LOC).

5. Расходы по статье «Машинное время» (*Рм*) включают оплату машинного времени, необходимого для разработки и отладки программного продукта, которое определяется по нормативам (в машино-часах) на 100 строк исходного кода (*Hмв*) машинного времени, и определяются по формуле:

(4.8)

где Цм - цена одного машино-часа. Рыночная стоимость машино-часа компьютера со всем необходимым оборудованием (0,9 BYN / ч);

Vо – уточнённый общий объём функций строк исходного кода (LOC);

Нмв – норматив расхода машинного времени на отладку 100 строк кода, машино-часов. Принимается в размере 0,8.

6. Расходы на научные командировки (Рнк) определяются либо из сметы научных командировок, разрабатываемой на предприятии, либо в процентах от основной заработной платы исполнителей (10-15%).

7. Расходы по статье «Прочие затраты» (Пз) на ПС включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы и определяются по формуле

 (4.9)

где Нпз – норматив прочих затрат в целом по организации, 15%.

8. Затраты по статье «Накладные расходы» (Рн) определяются по формуле:

 (4.10)

где Нрн – норматив накладных расходов в целом по организации, 50%.

9. Общая сумма расходов по всем статьям сметы (Ср) на ПС рассчитывается по формуле

 (4.11)

где ЗПо – основная зарплата, рассчитывается по формуле (4.3);

ЗПд – дополнительная зарплата, рассчитывается по формуле (4.5);

Рсоц – социальные расходы, рассчитывается по формуле (4.6);

Рс – расходы по статье «Спецоборудование»;

Рм – расходы по статье «Машинное время», рассчитывается по формуле (4.7);

Рнк –расходы на научные командировки, рассчитывается по формуле (4.8);

Пз– расходы по статье «Прочие затраты», рассчитывается по формуле (4.9);

Рн – затраты по статье «Накладные расходы» рассчитывается по формуле (4.10).

10. Затраты на освоение ПП(Ро). Организация-разработчик участвует в освоении ПП и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Для упрощения расчётов затраты на освоение определяются по установленному нормативу от общей суммы расходов по всем статьям сметы:

Ро = Ср \* Но /100, (4.12)

где Но – норматив затрат на освоение, 10%.

Ср – сумма расходов, BYN.

11. Затраты на сопровождение Рсо. Организация-разработчик осуществляет сопровождение ПП и несёт расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Для упрощения расчётов определяются по установленному нормативу от общей суммы расходов по всем статьям сметы:

Рсо = Ср \* Нсо/100. (4.13)

где Нco – норматив затрат на сопровождение, 10%.

12. Полная себестоимость рассчитывается по следующей формуле:

(4.14)

где Ср – сумма расходов, рассчитывается по формуле;

Ро – затраты на освоение, рассчитывается по формуле;

Рсо – затраты на сопровождение, рассчитывается по формуле.

Результаты расчета полной себестоимости программного обеспечения приведены в таблице 4.3.

Таблица 4.3 – Расчёт полной себестоимости программного обеспечения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование | Норматив | Расчетная формула | Сумма затрат, BYN |
| A | B | C | D |
| 1 | Зарплата, всего | - | - | 227,0 |
| 1.1 | Основная зарплата | - | - | 195,7 |
| 1.2 | Дополнительная зарплата | - | - | 31,3 |
| 2 | Отчисления на социальные нужды | 34,6 % |  | 78,5 |
| 3 | Спецоборудование | Не применялось | | - |
| 4 | Материалы | Не применялось | | - |
| 5 | Машинное время | - |  | 27,9 |
| 6 | Научные командировки | Не планировались | | - |
| 7 | Прочие затраты | 15% |  | 29,4 |
| 8 | Накладные расходы | 50% |  | 97,85 |
| 9 | Сумма затрат | - | 195,7 + 31,3 + 78,5 + 27,9 + 29,4 + 97,9 | 460,7 |
| 10 | Затраты на освоение ПП | 10% |  | 46,07 |
|  | Наименование | Норматив | Расчетная формула | Сумма затрат, BYN |
| A | B | C | D |
| 11 | Затраты на сопровождение | 10% |  | 46,07 |
| 12 | Полная себестоимость | - |  | 552,84 |

Общая сумма расходов на разработку как полная себестоимость программного средства (Сп) равна 552,84 BYN.

## 4.3 Расчет цены и прибыли по программному продукту

Для определения цены ПП необходимо рассчитать плановую прибыль, которая рассчитывается по следующей формуле:

, (4.15)

где Сп – полная себестоимость программного продукта, BYN;

R – уровень рентабельности ПП, 30%.

После расчета прибыли от реализации определяется прогнозируемая цена ПП без налогов:

, (4.16)

где П – плановая прибыль от реализации ПО, BYN.

Отпускная цена (цена реализации) ПП включает налог на добавленную стоимость и рассчитывается по формуле:

, (4.17)

где НДС - налог на добавленную стоимость определяется по формуле (4.18).

, (4.18)

где Цп – прогнозируемая цена, бел. руб.;

НДС - налог на добавленную стоимость, 20%.

Прибыль от реализации ПП за вычетом налога на прибыль (Пч) является чистой прибылью, остается организации-разработчику и представляет собой экономический эффект от создания нового программного продукта:

, (4.19)

где П – плановая прибыль от реализации ПО, BYN;

Нп – ставка налога на прибыль (в настоящее время Нп = 18%).

Проведем расчет отпускной цены ПО, результаты занесем в таблицу 4.4.

Таблица 4.4 – Расчет отпускной цены ПО

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование статей затрат | Норматив | Расчетная формула | Сумма затрат, BYN. |
| A | B | C | D |
| 1 | Полная  себестоимость | - | - | 552,84 |
| 2 | Прибыль | 30% |  | 165,85 |
| 3 | Цена без НДС | - | 552,84+165,85 | 721,69 |
| 4 | НДС | 20% |  | 144,3 |
| 5 | Отпускная цена | - |  | 866 |
| 6 | Чистая  прибыль | 18% |  | 136 |

Были определены основные экономические показатели:

1. Полная себестоимость – 552,84 BYN;
2. Прогнозируемая цена – 866 BYN;
3. Чистая прибыль – 136 BYN.

Таким образом, рассчитанная отпускная цена на программный продукт, разрабатываемый в рамках данного дипломного проекта, является конкурентоспособной. Согласно различным источникам, текущая рыночная цена на подобный программный продукт (лицензия на год) в Республике Беларусь колеблется в диапазоне от 1300 BYN до 3000 BYN.

# ЗАКЛЮЧЕНИЕ

В рамках дипломного проекта был изучен ОА, было разработано программное обеспечения для учета ПЭВМ и оргтехники для ГУ Минфина РБ по Брестской области.

В ходе выполнения дипломного проектирования был проведен системный анализ объекта автоматизации. После полного изучения структуры объекта автоматизации описаны функции и поставлены задачи объекта автоматизации.

Определена целесообразность автоматизации и выполнена постановка задачи на создание системы.

Была спроектирована структура системы, после чего разработана концептуальная, логическая и физическая модель базы данных. Полностью описана и реализована структура пользовательского интерфейса.

Выполнен расчет экономических показателей, итоговая отпускная цена получилась равной 9 305 071,37 бел. руб., чистая прибыль составила 1 467 338,18 бел. руб.

Были созданы документы по установке ПО, проведено успешное испытание системы.

# СПИСОК СОКРАЩЕНИЙ

АС – автоматизированная система.

БД – база данных.

ГОСТ – государственный стандарт.

ГУ – главное управление.

ДП – дипломный проект.

ОА – объект автоматизации.

ПИ – пользовательский интерфейс.

ПО – программное обеспечение.

ПС – программная система.

СУБД – система управления базами данных.

ТЗ – техническое задание.

# СПИСОК ЛИТЕРАТУРЫ

1. ЕСПД. Техническое задание. ГОСТ 19.201-1978.
2. ЕСПД. Текст программы. ГОСТ 19.401-1978.
3. ЕСПД. Программа и методика испытаний. ГОСТ 19.201-2000.
4. ЕСПД. Автоматизированные системы. Стадии создания. ГОСТ 34.003-1992.
5. ЕСПД. Схемы алгоритмов, программ, данных и систем. ГОСТ 19.701-1990.
6. Кулакова Л.О., Кичаева Т.В. Методические указания по технико-экономическому обоснованию дипломных проектов для студентов специальности АСОИ. БрГТУ: Брест, 2015.
7. Мейер Э.А. Каскадные таблицы стилей. Подробное руководство/Э.А. Мейер. – СПб., 2008. – С. 385–418.
8. Основные функции Javascript [Электронный ресурс] – 2015 – Режим доступа: http://javascript.ru/tutorial. Дата доступа: 29.04.2015.
9. Основы энергосбережения [Электронный ресурс] – 2015 – Режим доступа: http://old.computerra.ru/notebooks/583207. Дата доступа: 20.04.2015.
10. Полное руководство Yii Framework и официальная документация [Электронный ресурс] – 2015 – Режим доступа: http://yiiframework.ru/doc/cookbook. Дата доступа: 05.04.2015.
11. The JQUERY plugin registry [Электронный ресурс] – 2015 – Режим доступа: http://plugins.jquery.com. Дата доступа: 16.04.2015.