Федеральное государственное бюджетное образовательное учреждение   
высшего образования

|  |  |
| --- | --- |
| Gerb-BMSTU_01 | «Московский государственный технический университет  им. Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана) |

ФАКУЛЬТЕТ – Информатика и управления

КАФЕДРА – Информационные системы и телекоммуникации

Курсовая работа

на тему

Разработка системы управления терминалами

Студент группы ИУ3-71 (подпись) 17.09.2021 А.Т. Марчук

Руководитель курсовой работы (подпись) 17.09.2021 И.М. Сидякин

Москва, 2021

Содержание

# Введение

На сегодняшний день остро стоит вопрос безналичных платежей. С каждым годом их востребованность растет, как и растет потребность в системах удаленного контроля, мониторинга и наладки для терминалов безналичных платежей. Для упрощения данного процесса осуществляют разработку так называемых TMS – Terminal Management System – системы управления терминалами. Данные системы должны осуществлять мониторинг состояния терминалов и позволять удаленно настраивать их.

На сегодняшний день существует множество различных систем управления терминалами от различных производителей. Возникает закономерный вопрос – так ли остро стоит вопрос разработки еще одной? Проблема заключается в том, что имеет место быть сильная зависимость между аппаратной составляющей терминала и тем, какая система управления может быть к нему применена. Потребность разработки собственной системы управления обусловлено несовместимостью изготавливаемых терминалов с теми системами управления, которые в настоящий момент есть на рынке. Однако для понимания характерных признаков данных систем стоит привести выборку из нескольких, наиболее популярных, и проанализировать их преимущества и недостатки.

# Исследовательская часть

Настоящий раздел является теоретическим и включает:

* анализ требований технического задания;
* описание предметной области разработки;
* анализ существующих решений в данной предметной области;
* выбор и обоснование технического решения.

## Анализ требований технического задания

/\*ВСТАВИТЬ СКАН ПОДПИАННОГО ЗАДАНИЯ\*/

## Описание предметной области разработки

Введем несколько определений и аббревиатур для дальнейшего использования.

EMV – Стандарт взаимодействия типовых карт для проведения платежей по карте

EMV приложение – Микропрограмма и блок параметров на карте, относящейся к международной платежной системе EMV

ПИН-пад – Электронное устройство, предназначенное для ввода PIN-кода

PIN код – Секретный код, вводимый клиентом, для подтверждения того, что он обладает полномочиями по использованию карты

Терминал – Аппаратно-программный комплекс, предназначенный для регистрации факта приобретения товара и упрощения процедуры приобретения товара

Код действия терминалов (TAS) – Действия ПО терминала при возникновении конкретной ситуации (отказ от операции, проведение операции онлайн или действие по умолчанию)

Коммуникационный скрипт – Минипрограмма (последовательность команд), определяющая сценарий подключения терминала к серверу авторизации или серверу TMS

Клиент/Держатель – Физическое лицо (в том числе уполномоченный представитель юридического лица), на имя которого в силу заключенного между ним и Эмитентом договора и/или на имя которого по указанию лица, заключившего договор с Эмитентом, выпущена Карта.

Магнитная полоса - Полоса, содержащая идентификационную информацию о карте и ее держателе.

Операция - Расчетная операция, осуществляемая с использованием Карты с обязательной Авторизацией в соответствии с законодательством Российской Федерации, нормативно-правовыми актами Банка России и договором с Эмитентом, проводимая в Торговой точке по требованию Держателя.

Операция возврата - Расчетная операция, совершаемая с использованием Карты при возникновении необходимости в возврате Держателю денежных средств, списанных с его счета в рамках ранее совершенной Операции.

Операция отмены - Процедура отмены Кода Авторизации до формирования Электронного журнала.

Электронный журнал - Документ или совокупность документов в электронной форме, сформированные за определенный период времени при совершении Операций/Операций с использованием POS-терминала.

Программное обеспечение (ПО) - Программное обеспечение POS-терминала.

Профиль - Один из объектов системы, предназначенный для хранения настроек групп параметров, определяющих алгоритм работы терминала в процессе выполнения операции с картой или административной операцией.

Реквизиты Карты - Номер Карты, срок действия Карты, код безопасности CVV2/CVC2/Batch code/СVV II/CAV2 (для Карт Платежных систем Visa International/MasterCard Worldwide/American Express/Diners Club International/JCB International соответственно).

Сервисный код карты - Код карты, который указывается область ее действия и тип.

Сервисный код коммерсанта (MCC) - Четырехзначный номер, который обозначает профиль деятельности, которым занимается коммерсант.

Скрипт эмитента - Команда или командная строка, передаваемая от эмитента к терминалу с целью выполнения команд(ы) микропроцессорной картой.

Способ верификации владельца карты - Способ верификации того, что человек, предъявляющий карту, имеет права на ее использование (например, сверка подписи или введение PIN кода).

Справочник - Один из объектов системы, предназначенный для хранения основных

значений, шаблонов, терминов и алгоритмов, которыми оперирует система.

Транзакция - Группа последовательных операций, которая представляет собой логическую единицу работы с данными. Транзакция может быть выполнена либо целиком и успешно, соблюдая целостность данных и независимо от параллельно идущих других транзакций, либо не выполнена вообще и тогда она не должна произвести никакого эффекта. Транзакции обрабатываются транзакционными системами, в процессе работы которых создаётся история транзакций.

Хост, Коммуникационный сервер - Сервер базы данных платежной системы, который отвечает на запросы POS-терминала точки обслуживания, а также снабжает его необходимой информацией.

Эквайер - Банк или компания, осуществляющая весь спектр операций по взаимодействию с точками обслуживания карточек, которая состоит из терминалов в торгово-сервисной сети и банкоматов.

Эмитент - Организация, выпустившая (эмитировавшая) карточки для развития и

финансирования своей деятельности

AID - Application Identification Number (Идентификационный номер приложения)

AVN - Application Version Number (Номер версии приложения)

BIN - Bank Identification Number (Банковский идентификационный номер)

CAP - Cash register Application

CVM - Cardholder Verification Method (Способ верификации владельца карты)

DDOL - Dynamic Data Object List (Список элементов данных необходимых для динамической аутентификации)

ICC - Integrated Circuit(s) Card (Карта с интегрированными электронными схемами)

ID - Identifier (идентификатор)

ISO - International Organization for Standardization (Международная организация стандартизации)

MCC - Merchant Category Code (Сервисный код коммерсанта)

MS - Magnetic Stripe (магнитная полоса)

TAC - Terminal Action Code(s) (Код(ы) действия терминалов)

PAN - Primary Account Number (Индивидуальный номер карты)

PIN - Personal Identification Number (Персональный идентификационный номер)

POS - Point of Sale (точка продажи, сервиса)

RID - Registered Application Provider Identifier (Зарегистрированный номер провайдера приложения; например, VISA, MasterCard)

TDOL - Transaction Data Object List (Список данных о транзакции участвующих при вычислении сертификата)

TMM - Terminal Management Mode (модуль управления терминалами)

TMS - Terminal Management System (система управления терминалами)

СУБД - Система управления базами данных

Современные POS-системы представляют собой комбинацию программных и аппаратных решений, позволяющих проводить платежные операции и облегчающих ежедневные бизнес-процессы. Говоря о POS-ах, обычно имеют в виду кассовые аппараты, терминалы оплаты и другие привычные составляющие торговых магазинов. Однако, архитектура POS не ограничивается только этими элементами. Пример полной процедуры транзакции приведен на рисунке 1.



Рисунок 1 – пример полной процедуры транзакции

Сначала держатель проводит картой по считывателю. Данные кредитной карты поступают в терминал, откуда отправляются в POS-систему. Далее, POS-система связывается сервисом обеспечения платежей, который, в зависимости от типа кредитной карты, обращается в банк для прохождения процедуры авторизации транзакции. В этот момент держателю предлагается ввести PIN-код для подтверждения транзакции. Если все прошло успешно, код авторизации возвращается из банковской сети в PSP и передается в POS-систему и терминал.

В рамках данной дипломной работы будет осуществлены анализ и доработка системы, осуществляющей работу с серверной частью По и позволяющей осуществлять мониторинг и конфигурирование удаленных терминалов.

## Анализ существующих решений в данной предметной области

Для анализа было выбрано 3 реализации TMS – XENTURION, OtiTms и ARCUS2. Ввиду невозможности исследования проприетарного исходного кода, рассмотрим список особенностей, которые они предоставляют. Последовательно проанализируем документацию каждой из систем управления терминалами, выделяя отличительные черты.

XENTURION предлагает следующий функционал:

* Удаленное управление терминалами
* Организация и настройка набора терминалов
* Использование мощного поискового движка для установления работающего списка терминалов
* Просматривать статус терминалов
* Производить обновления в 1 клик
* Просматривать историю действий
* Загружать произвольное программное обеспечение в терминалы
* Планировать умные и динамические обновления в нерабочие часы
* Управлять загрузками для различных семейств терминалов
* Просматривать прогресс обновлений
* Загружать и выгружать любой параметризованный объект с терминала на сервер
* Доставлять начальную конфигурацию на сервер
* Разделять параметры между группой терминалов
* Просмотреть обновления параметров
* Устанавливать повестку для синхронизации сервисов
* Планирование включений и отключений
* Ограничивать доступ для партеров с ограниченным допуском
* Просматривать историю для каждого из пользователей
* Упрощение удаленное распределение ключей и сертификатов для терминалов
* Просматривать историю изменения ключей

В то же самое время, список преимуществ OtiTms следующий:

* Управление финансовой, технической и продуктовой информацией
* Мониторинг аппаратной материальной базы
* Детализированный технический мониторинг
* Настраиваемое расписание уведомлений
* Мониторинг и оповещение продаж, как наличных, так и безналичных
* Торговая аналитика по каждому клиенту/местоположению/терминалу
* Удаленная смена настроек конфигурации.

Список отличительных черт ARCUS2:

* Высокая скорость совершения транзакций
* Длительная наработка на отказ
* Работа в режиме многозадачности
* Стандартная поддержка не только магнитных лент, но и бесконтактных карт
* Множественность каналов связи для подключения

Резюмирую, можно сказать, что для TMS являются ключевыми следующие параметры:

* Возможность удаленного мониторинга и настройки
* Высокая степень оптимизации
* Возможность контроля доступа и логгирования

## Выбор и обоснование технического решения

В рамках данного проекта мною будет разрабатываться веб-приложение с использованием фреймворка для серверной разработки Spring Boot, фреймворка для разработки клиентской части ReaсtJS, а так же с использованием СУБД MySQL. Данный выбор был обусловлен тем, что данные решения являются востребованными на рынке в настоящее время, хорошо масштабируются и подходят для решения поставленных задач. Кроме того, определенная доля разрабатываемой системы управления терминалами была разработана предварительно с использованием данного перечня технологий, ввиду чего мной было принято решение оставить данный стек до завершения проекта. Кроме того, на финальных этапах разработки потребуется сервис для помещения всех терминалов в одну локальную сеть. Для этого будет использоваться wireGuard.

# Конструкторская часть

В рамках данной курсовой работы были предприняты следующие действия:

Разработана система логгирования событий, расширяющая и дополняющая существовавшую логгирующую систему:

Переработана система получения терминалам информации об изменении конфигурационных файлов, а также доработана система сбора статистики, позволяющая получать данные не по инициативе оконечного терминала, а посредством запроса с сервера, что существенно расширяет существующий функционал.

Разработан файл конфигурации образа виртуальной машины, позволяющий быстро кроссплатформенно разворачивать сервер с программным продуктом, а также использовать специализированные решения для автоматической балансировки нагрузки в высоконагруженных системах.

Разработано кроссплатформенное приложение для мобильных устройств, служащее для тестирования и отладки серверной части приложения, а также – имеющее потенциал для последующего преобразования в мастер-терминал.

Ниже будет рассмотрен каждый из вышеописанных этапов.

## Система логгирования событий.

Ввиду того, что основное ядро программного продукта написано в виде монолитного сервиса на языке Java, для снижения сложности поддержки системы в целом, упрощении ее дальнейшей поддержки и возможности постепенного перехода на микросервисную архитектуру, логгирование осуществляется посредством написания отдельного программного продукта, придерживающегося паттерна построения «микросервис». Данный микросервис осуществляет только одну задачу – принимает запрос от некоторого сетевого хоста и обрабатывает его. В частности, в роли сетевого хоста выступает сервер фронтенда, а в качестве результата обработки запроса можно назвать добавление новой записи в таблице логов в базе данных. Таблица логов в базе данных не является сильно связанной с остальными таблицами базы данных. Поэтому добавление в нее записей можно вынести в отдельную зону ответственности, что и было сделано путем разработки микросервиса логгирования.

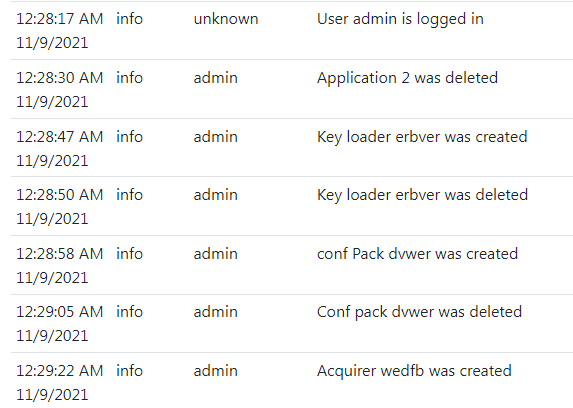


Рисунок [todo вставить номер рисунка] – пример отображения записей из таблицы логов

В качестве языка разработки для данного микросервиса был выбран NodeJS. Причиной этому послужили множественные факторы, такие как простота разработки и поддержания кода, написанного на данном языке, тот факт, что NodeJS является актуальным и динамично развивающимся языком, который нашел свое применение в множественных отраслях разработки серверных приложения, а также крайне большой ассортимент необходимых библиотек. В частности, библиотека mysql2 позволяет гибко настраивать произвольные запросы к базе данных посредством инъекций переменных в предопределенные структуры на языке SQL, избегая излишних абстракций, что, в свою очередь, приводит к повышению производительности, а также позволяет участвовать в разработке проекта лиц, в не зависимости от их умения использования языки серверной разработки, например, инженеров баз данных. Формат запроса к микросервису также претерпел изменения по сравнению с ядром. Если в основной программе серверного приложения используется устаревший формат передачи данных XML, то в микросервисах, дописываемых поверх него, вся логика взаимодействия микросервисов построена на использовании формата JSON, больше распространенного в современной WEB разработке. Конечно, стоит отметить, что парсинг данных не производится вручную – анализирование поступающей информации осуществляется с помощью набора из нескольких специализированных библиотек, которые позволяют абстрагироваться от сетевого уровня действия протоколов и работать с поступающими данными так, словно они поступают не с терминала, удаленного, возможно, на многие и многие мили, но - хранятся на том же устройстве, что и код, их обрабатывающий.

Однако, помимо общего описания стека технологий, требуется также описать логику обработки данных, использующуюся в данном микросервисе. В рамках данного микросервиса происходит создание соединения с базой данных, определяющей параметры подключения, такие как используемая учетная запись, пароль и хост. В общем случае, подобные данные должны быть получаемы из переменных окружения, однако в демонстрационных целях в рамках данной курсовой работы было принято решение вставить их непосредственно в текст программы.

Затем определяется особый объект, называемый приложением. От определяет используемый порт – в данном случае был выбран порт 8081 – используемый тип кодирования и утилиту для расшифровки JSON, текущую политику защиты от междоменных запросов и подобные параметры. Затем для данного приложения определяется один обрабатываемый путь - в частности, это приложение может принимать POST запросы с путем /api/log, причем данные запросы должны содержать в своем теле поля date, level, user и message. Стоит отметить, что поле data является опциональным, так как имеет определенный смысл использовать для логгирования не клиентское время пользователя, эксплуатирующее систему, а серверное время, единое для всех терминалов. С другой стороны, нельзя не отметить, что в терминале булл реализован определенный функционал, которые не будет позволять пользоваться сервером хостам, чье отставание или опережение по времени находится за пределами определенной, допустимой погрешности. Данный прием осуществляется главным образом из соображений безопасности, но нельзя не заметить, что корректность выставления времени на каждом из устройств является необходимым условием поддержания порядка в системе, что чрезвычайно важно для любого информационного продукта в целом и для системы управления банковскими терминалами в частности.

Параметр date определяет время, в которое произошло логгируемое событие. Level определяет уровень важности произошедшего события – оно может быть информационным, предупреждающим или критическим. Однако, стоит отметить, что, ввиду того, что поле, принимающее данный параметр, является строковым, множество различных уровней можно крайне существенно расширить. Параметр message определяет некоторое сообщение, которое может быть опционально передано в систему и характеризующее произошедшее событие, например, авторизация пользователя, удаление, добавление или изменение каких либо записей. User определяет псевдоним пользователя. Чьи действия вызвали процедуру логгирования. В случае, если процедура логгирования не была связана с пользователем, в данном поле предполагается указывать некоторое зарезервированное слово, например System.

Кроме того, в рамках данного микросервиса определена функция, которая принимает на вход вышеописанные параметры, преобразует их аргументы для команды INSERT языка SQL и передает в базу данных в виде строкового текста. Затем дожидается ответа, стрингифицирует его и передает в качестве ответного запроса на входящий запрос вопрошающей стороне. Текст кода программы предоставлен в репозитории Github. Ссылки на Github приведены в приложении В.

## Микросервис сокетов

Нельзя преумалять важность логгирования действий, происходящих в системе. Именно подробные логи позволяют анализировать происходящие в системе процессы, определить зоны ответственности пользователей или – определить, что послужило причиной той или иной ошибки или сбоя. Однако в рамках данной курсовой работы не меньшего внимания заслуживает и микросервис сокетов.

Данное называние не совсем корректно, так как оно в большей степени отражает технологию реализации процессов, осуществляемых данным сервисом, нежели задачи, которые данный сервис решает, однако ввиду отсутствия сколь-либо приемлемых альтернатив – данное название можно назвать подходящим, по крайней мере в рамках рабочей документации.

Для описания решаемых задач стоит рассмотреть два подхода к доставке информации на удаленные терминалы. Сразу стоит оговориться, что терминалы являются пассивными устройствами, они могу высылать запросы на сервер, однако не могут определять сложную логику рассылки данных запросов. С учетом вышесказанного, очевидными являются только 2 варианта получения терминалами информации – это либо регулярный опрос сервера, либо использование запросов, высылаемых с сервера. Первый вариант был реализован в ранних стадиях реализации программного продукта, однако оказался плох, ввиду своей низкой производительности, невозможности работать с большим количеством устройств, а также чрезмерным потреблением интернет-канала. Второй способ можно разбить на 2 варианта – можно либо использовать сервер на стороне терминала, посылая на него POST запросы, либо организовать прямое соединение по типу сокета. Каждый из них имеет свои преимущества и недостатки, однако соединение посредством сокета оказывается более предпочтительным, так как его преимущества являются более весомыми. В частности, оно потребляет меньше ресурсов на терминале, может быть активировано как терминалом, так и сервером и имеет крайне малое время отклика ввиду отсутствия необходимости установки соединения. С другой стороны, данный подход ограничивает количество подключаемых терминалов значением 65534 на один белый IP, чего в общем случае должно быть достаточно, особенно учитывая ограниченные вычислительные мощности сервера.

В данном контексте особого внимания заслуживает фреймворк Socket.IO. Данный фреймворк является комплексом высокоуровневых оберток над библиотекой WebSocket, что позволяет не реализовывать существенный объем требуемой логики вручную, а полагаться на программные решения, предварительно созданные разработчиками. В частности, в отличии от вебсокетов, сокет.ио поддерживает восстановление соединения после кратковременного разрыва, позволяет использовать вебсокеты посредством альтернативных каналов связи, поддерживают проксирования, что может оказаться критически важным при необходимости работы в нескольких сетях, а также поддерживают изящную деградацию, что в свою очередь обеспечивает простоту их внедрения даже на ранних стадиях написания продукта. Более детально вопросы проксирования будут рассмотрены ниже, когда речь пойдет о настройке среды и тотальной виртуализации системы, которая требуется для возможности передачи информации не только от терминалов к серверу, но и от сервера к терминалам по его инициативе, в условиях отсутствия возможности использования белого IP адреса для каждого из терминалов, ввиду низкой экономической эффективности данного технического решения.

Ключевой момент. Который стоит рассматривать, говоря о данном микросервисе, это именно обработка логики сокетов. Однако нельзя не упомянуть, что значительную часть объема кода занимает первичное конфигурирование приложения, а также ряд внутренних функций, например – позволяющих представлять аргументы строки запроса в виде объекта языка JavaScript. Настройка приложения в общем смысле аналогична настройке приложения для сервиса логгирования, поэтому акцентирования внимания на ней удастся избежать, однако пристальному рассмотрению должен подлежать объект TID\_connections и метод find\_TID. В объекте TID\_connections расположено множество объектов с полями, описывающими id соединения, объект соединения и идентификатор терминала. При первом подключении какого-либо устройства сервер проверяет, есть ли идентификатор терминала хотя бы одном из объектов, хранящихся в коллекции соединений, и игнорирует попытку соединения в случае наличия. При этом в случае отсутствия в данном массив объекта, попадающего под заданные критерии, этот объект создается и добавляется в массив. В результате, в коллекции хранится множество объектов, позволяющих сопоставлять открытое серверное соединение с физическим устройством, зарегистрированным в базе, посредством проверки равенства TID терминала и идентификатора, записанного в поле идентификатора в коллекции соответствия TID и соединений. Кроме того, помимо соединения отдельно хранится и ID соединения. С одной стороны, данная мера избыточна, с другой стороны – хранение идентификатора позволяет существенно упростить разработки и отладку приложения. В случае отсоединения устройства от сервера, например, в случае его отключения или длительного прерывания соединения (как уточнялось выше, при кратковременных перебоях интерната соединение восстанавливается автоматически) – запись о том, что терминал по данному сокету подключен из массива TID\_connections удаляется, и становится невозможным выслать на данный терминал какой-либо запрос. При этом генерируется соответствующий ответ в качестве response на тот запрос, который пытался передать информацию в сокет. Перечень команд, их текстовых описаний, сигнатур запросов, а также используемые URL приведены в приложении Б.

Безусловно, нельзя не отметить, что несмотря на высокую степень проработанности данного микросервиса, в нем должны быть привнесены определенные изменения. В частности, на данный момент нет проверки на допустимость той или иной команды, не осуществляется проверка подлинности идентификатора в базе данных и так далее. Однако стоит сказать, что данная версия реализации практического задания к курсовой работе не является окончательной, а представляет из себя скорее доказательство работоспособности концепта, который будет развиваться далее и достигнет своего апогея в рамках написания не курсовой, но дипломной работы. С одной стороны, нельзя не признать, что даже в рамках курсовой работы исполнитель должен осуществлять комплекс работ, который позволяет получить в той или иной степени завершенный продукт, подходящий для непосредственного использования, с другой стороны нельзя не согласиться, что объем работ, подлежащих выполнению, в рамках данного проекта столь колоссален, что даже с учетом того факта, что существенная часть программного кода была в том или ином виде написана заблаговременно, полноценная разверстка и реализация поставленной задачи требует многократно большее количество человеко-часов, чем то, которое отводится на выполнение курсового проекта. Текст кода программы предоставлен в репозитории Github. Ссылки на Github приведены в приложении В.

## Настройка VPN

Предполагается, что разрабатываемая система должна работать на множестве терминалов, разнесенных в различные локальные сети, при этом использование для каждого терминала отдельного публичного IP не является рациональным, так как их ограниченное число и данный подход не был бы экономически оправдан. Кроме того, данный подход неприменим для областей, где плохое покрытие интернетом и есть только один провайдер, так как у него может не хватить диапазона. В связи с этим имеется потребность как-либо расположить все терминалы в одной локальной сети. Казалось бы. Это невозможно из-за причин, описанных выше, однако существует подход, позволяющий решить данную проблему. Речь безусловно идет о VPN сервисах. Для понимания процессов, описываемых далее в рамках данной курсовой работы надлежит ввести некоторые пояснения.

VPN – это аббревиатура для понятия виртуальная частная сеть. Виртуальная частная сеть есть совокупность технологий, позволяющих обеспечить одно или несколько сетевых соединений (логическую сеть) поверх другой сети (например, Интернет).

Расшифровка названия: сеть – объединение нескольких устройств каким-либо видом связи, позволяющее обмениваться информацией. Виртуальная – неосязаемая, не физическая, то есть не важно, по каким именно каналам связи она проложена. Физическая и логическая топологии могут как совпадать, так и отличаться. Частная – в эту сеть не может войти посторонний пользователь, там находятся только те, кому разрешили доступ. В частной сети надо маркировать участников и их трафик, чтобы отличить его от остальной, чужой информации. Также в такой сети обеспечивается защита данных криптографическими средствами, попросту говоря, шифруется.

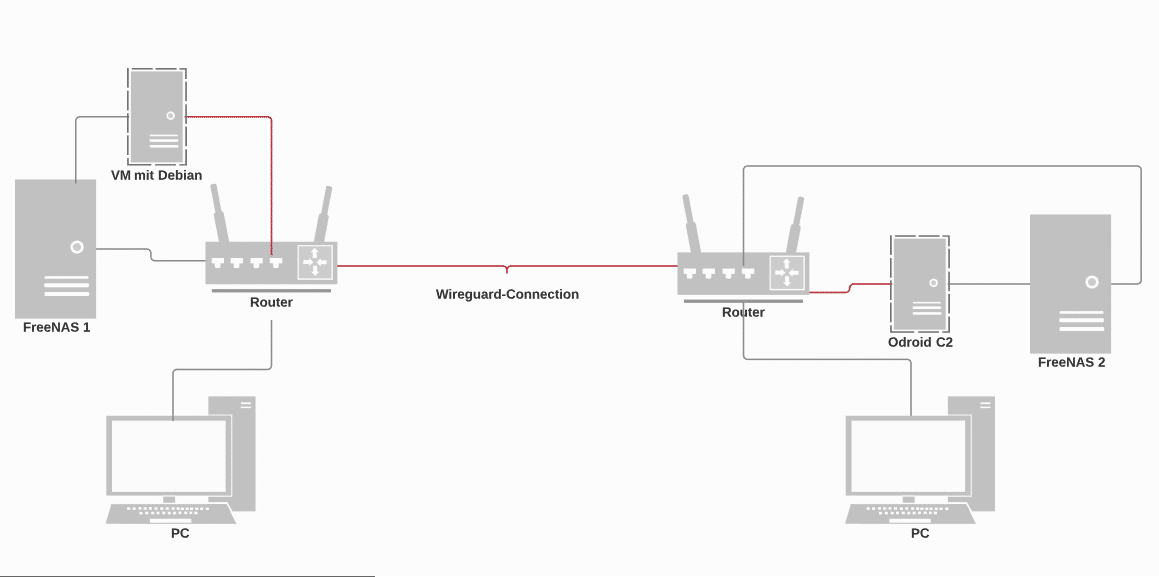


Рисунок [todo вставить номер рисунка] – принцип работы VPN туннеля

Таким образом, ввиду отсутствия необходимости публичного маркирования терминалов, достигается сразу несколько существенных преимуществ. Во первых, никакой наблюдатель извне сети не может определить ее структуру и топологию, а также не имеет информации о устройствах, сеть составляющих. Ввиду того, что работа идет в сфере, чуствительной к утечкам данных, это становится даже более важно, чем это есть в тривиальном контексте.

Безусловно, за любое преимущество приходится расплачиваться недостатками. В случае, с VPN сервисами это снижение скорости соединения, периодические кратковременные прерывания сети, невозможность использования сетей ipv6, протечки DNS и различного рода юридические сложности. Однако несмотря на большое количество недостатков, каждый из них по отдельности не является критичным для разрабатываемой системы. Проблема дефицита скорости не критична, так как сам подход к проектированию системы мониторинга не предполагает передачи по сети большого объема данных, чувствительного к скорости передачи. Прерывания подключения казалось бы могут быть критичными, так как в текущей реализации тестового стенда терминалы подключаются к серверу единожды, при загрузке, однако стоит вспомнить, что используемый фреймворк для сокетов SocketIO поддерживает автоматическое восстановление соединения, что позволяет бороться с данной проблемой даже без ведома конечного пользователя. DNS протечки не применимы в текущей системе просто по причине того, что при отправке запроса DNS сервер не задействуется – адресация идет непосредственно посредством IP адресов. Невозможность использования ipv6 все больше и больше уходит в прошлое и на сегодняшний день большая часть серверов постепенно переходят на его поддержку, что дает основания полагать, что к моменту введения системы в эксплуатацию, данная проблема будет некоторым анахронизмом и окончательно уйдет в прошлое.

Как бы странно ни было, но главной проблемой на текущий момент можно назвать юридические нюансы. Летом 2021ого года было подписано постановление, существенно ограничивающее возможности использования VPN, однако на момент написания данной работы полного запрета на подобные сервисы еще не было введено.

Как бы то ни было, было принято решение в данной работе использовать сервис VPN. На сегодняшний день существует большое количество приложений для этого, однако в рамках настоящей работы используется Wireguard. В общей смысле это не совсем средство VPN, ввиду того, что его возможности ограничены поддержанием предопределенного туннеля, однако в сочетании с другими программными решениями он выполняет поставленные перед ним задачи.

Официально заявляемые преимущества weireguard:

* Современные алгоритмы шифрования
* Высокая скорость работы
* Простая настройка и развёртывание
* Кроссплатформенность
* Защита от подделки ключей
* Защита от атак повторного воспроизведения

Кроме того, можно отметить, что wireguard внедряется в качестве нативного модуля линукса, из чего можно сделать вывод о том, что в течении времени его конфигурация станет еще проще.

Существует несколько подходов к развертыванию тех или иных утилит. Либо прямая

установка из установочного файла, при этом не принципиально, был ли данный файл скачан откуда-либо или изготовлен самостоятельно, и развертывание виртуальной машины с предзаготовленным контейнером. Хотя в рамках данной работы и будет использоваться развертка из контейнера, стоит упомянуть последовательность его работы. Непосредственно конфигурационные файлы будут прикреплены в приложения, в данном описании либо отмечены ключевые шаги вкупе с описанием реализаций тех или иных преимуществ.

Создается WireGuard интерфейс, ему назначается приватный ключ и IP адрес. Загружаются настройки других пиров: их публичные ключи, IP адреса и т.д.

Все IP пакеты, приходящие на WireGuard интерфейс инкапсулируются в UDP и [безопасно доставляются](https://www.wireguard.com/protocol/) другим пирам.

Клиенты задают публичный IP адрес сервера в настройках. Сервер автоматически узнает внешние адреса клиентов, когда от них приходят корректно аутентифицированные данные.

Сервер может менять публичный IP адрес не прерывая работы. При этом он отошлет оповещение подключенным клиентам и они обновят свою конфигурацию на лету.

Используется концепт маршрутизации [Cryptokey Routing](https://www.wireguard.com/#cryptokey-routing). WireGuard принимает и отправляет пакеты на основании публичного ключа пира. Когда сервер расшифровывает корректно аутентифицированный пакет, проверяется его src поле. Если оно соответствует с конфигурацией allowed-ips.

Как можно видеть, данная схема в совершенстве подходит под поставленные задачи.

Полный текст файла конфигурации приведен в приложении А.

Для корректной работы терминала с сервером требуется настройка не только серверного, но и клиентского приложения. Стоит добавить важную пометку – здесь и далее будет идти речь о конфигурировании VPN для устройства, эмулирующего терминал, а не непосредственно терминала, поэтому некоторые настройки могут в той или иной степени отличаться, а некоторые устанавливаемые пакеты программного обеспечения могут быть недоступны для установки на терминал. Однако базовый принцип сохраняется вне зависимости от целевой платформы.

1. Таким образом, для подключения устройства, имитирующего терминал к серверу посредством VPN туннеля, надлежит предпринять следующие действия:
2. Установить клиент WireGuard и запустить его
3. На экране внизу нажать на кнопку с символом «+», а затем выбрать создание вручную
4. В поле «Имя» ввести произвольное имя подключения
5. В поле «Закрытый ключ» ввести заранее изготовленный приватный ключ или сгенерировать ключ посредством нажатия на кнопку «Генерировать»
6. Поле открытый ключ должно быть заполнено автоматически при генерации закрытого ключа. В случае, если ключ вводился вручную, требуется введение и открытого ключа
7. В поле «Адрес» надлежит указать глобальный публичный адрес сервера. В случае с тестовой конфигурацией это 91.207.170.27
8. В поле «Порт» надлежит указать номер используемого физического номера порта, в частности 51820 для текущей конфигурации
9. После этого требуется сохранить созданный туннель, посредством нажатия на пиктограмму дискеты в правом верхнем углу
10. После сохранения, требуется подключиться к тоннелю. Для этого надлежит выбрать созданный туннель и передвинуть ползунок в правое положение.

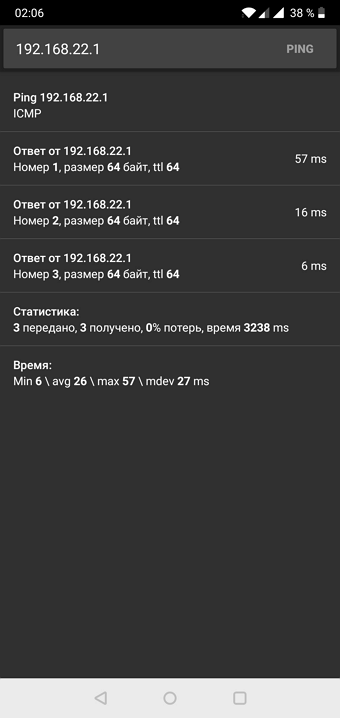
Для проверки работоспособности можно отправить ICMP пакет до сервера, обращаясь к нему по адресу в локальной сети, используя утилиту PingTool Network Utility.  


Рисунок [todo вставить номер рисунка] – Подтверждение наличия возможности обраться к удаленному серверу через VPN соединение.

## Разработка конфигурации для сервисов виртуализации

В настоящее время существенная часть программных продуктов не запускается на сервере напрямую, а делает это используя программные средства для виртуализации данного процесса. Разрабатываемое в рамках данной работы серверное приложение не стало исключением – при его развертывании тоже используется виртуализация.

Для того, что бы раскрыть необходимость данного подхода, предварительно необходимо сказать пару слов о том, что такое виртуализация и какие преимущества и недостатки она дает. В общем смысле виртуализация есть представление набора вычислительных мощностей вне зависимости от их аппаратно реализации, посредством добавления дополнительного уровня абстракции, и обеспечивающее при этом логическую независимость друг от друга нескольких параллельно идущих процессов.

В данном исследовании нет большого смысла углубляться в тонкости и рассматривать различные виды виртуализации, такие как виртуализации приложений, серверов, представлений и так далее. Все они в той или иной степени имеют что-то общее, однако помимо этого имеют ряд малозначительных в текущем контексте различий. На текущий момент более важным будет сконцентрироваться на преимуществах, которые дает виртуализация приложению.

Главное преимущество, важность которого кратно возрастает в исследуемой предметной области – это безопасность. Приложение, запущенное в изолированной виртуальной среде не способно никаким образом, кроме заранее и явно определенных, взаимодействовать с другими приложениями, а это в свою очередь гарантирует, что данные сервера не будут повреждены ни умышлено, ни случайно. Более того, если в одном из потоков виртуализации произойдет ошибка, приводящая в случае отсутствия виртуализации к критической ошибке операционной системы, то в случае использования дополнительного уровня абстракции с учетом того, что он был корректно сконфигурирован прекратит работу только проблемный поток. А в случае, если настройка была расширенной – через некоторое время работа контейнера будет восстановлена автоматически. Из этого сразу же вытекает другое значительное преимущество подхода к развертыванию приложений на виртуальных машинах – их намного проще конфигурировать, а зачастую их достаточно сконфигурировать только 1 раз, а при переносе контейнера на другой сервер – проблемы могут возникнуть только при сильно различающихся настройках сервера исходного и целевого, как-то по разному открытые порты или разные права привилегий учетных записей. Помимо удобства поддержки из этого также можно сделать вывод о повышении скорости обновления – вместо того, что бы обновлять каждую из зависимостей вручную, достаточно обновить лишь один образцовый сервер, изменения с которого распространяться на остальные в полуавтоматическом режиме, что с вою очередь позволяет поддерживать версию сервера актуальной на каждом из них даже если отсутствует централизованное серверное хранилище, а физически сервера существенно разнесены, например, для различных территориальных округов. Еще один аспект, который можно считать доводом в пользу виртуализации это возможность запуска нескольких параллельных изолированных процессов на одном и том же устройстве. Это вторично для итогового продукта, однако критически важно в рамках тестирования, так как ввиду объективных причин во время разработки у автора не было возможности использования выделенного сервера.

Говоря о преимуществах виртуализации, нельзя не упомянуть и о ее недостатках. В частности, запуск виртуальной машины, даже в случае, если она сравнительно «тонкая» тоже потребляет ресурсы сервера, поэтому приложение, работающее в виртуальной среде при прочих равных будет работать несколько медленнее, чем оно же, работающее непосредственно на сервере. Кроме того, в данной работе это не актуально, так как в виртуальной среде действуют несколько другие правил лицензирования ПО, что в с одной стороны не является недостатком, но с другой – тоже не стоит упускать из внимания.

Однако, после того, как было рассмотрено, почему требуется использовать виртуализацию, следует определить, какой именно способ виртуализации использовать. Ввиду того, что подключение к серверу предполагается только посредством программного интерфейса, то есть пользователям никогда и ни при каких обстоятельствах не потребуется видеть непосредственно интерфейс среды, сразу исчезает необходимость в гипер терминалах, что влечет за собой использование сравнительно легкого как в настройке, как и в вопросе потребляемой мощности Docker.

Для того, чтобы определить оправданность его использования, стоит кратко рассмотреть что это такое и какой функционал предлагает. В частности, Docker это программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы; позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, а также предоставляет среду по управлению контейнерами. Ключевое преимущество Докера в том, что он позволяет пользователям упаковать приложение со всеми его зависимостями в стандартизированный модуль для разработки. В отличие от виртуальных машин, контейнеры не создают такой дополнительной нагрузки, поэтому с ними можно использовать систему и ресурсы более эффективно. Для понимания причин данного преимущества стоит более полно раскрыть разницу между Докером и классической виртуальной машиной.

Виртуальные машины отлично подходят для полной изоляции процесса для приложения: почти никакие проблемы основной операционной системы не могут повлиять на софт гостевой ОС, и наоборот. Но за такую изоляцию приходится платить. Существует значительная вычислительная нагрузка, необходимая для виртуализации железа гостевой ОС.

Контейнеры используют другой подход: они предоставляют схожий с виртуальными машинами уровень изоляции, но благодаря правильному задействованию низкоуровневых механизмов основной операционной системы делают это с в разы меньшей нагрузкой.

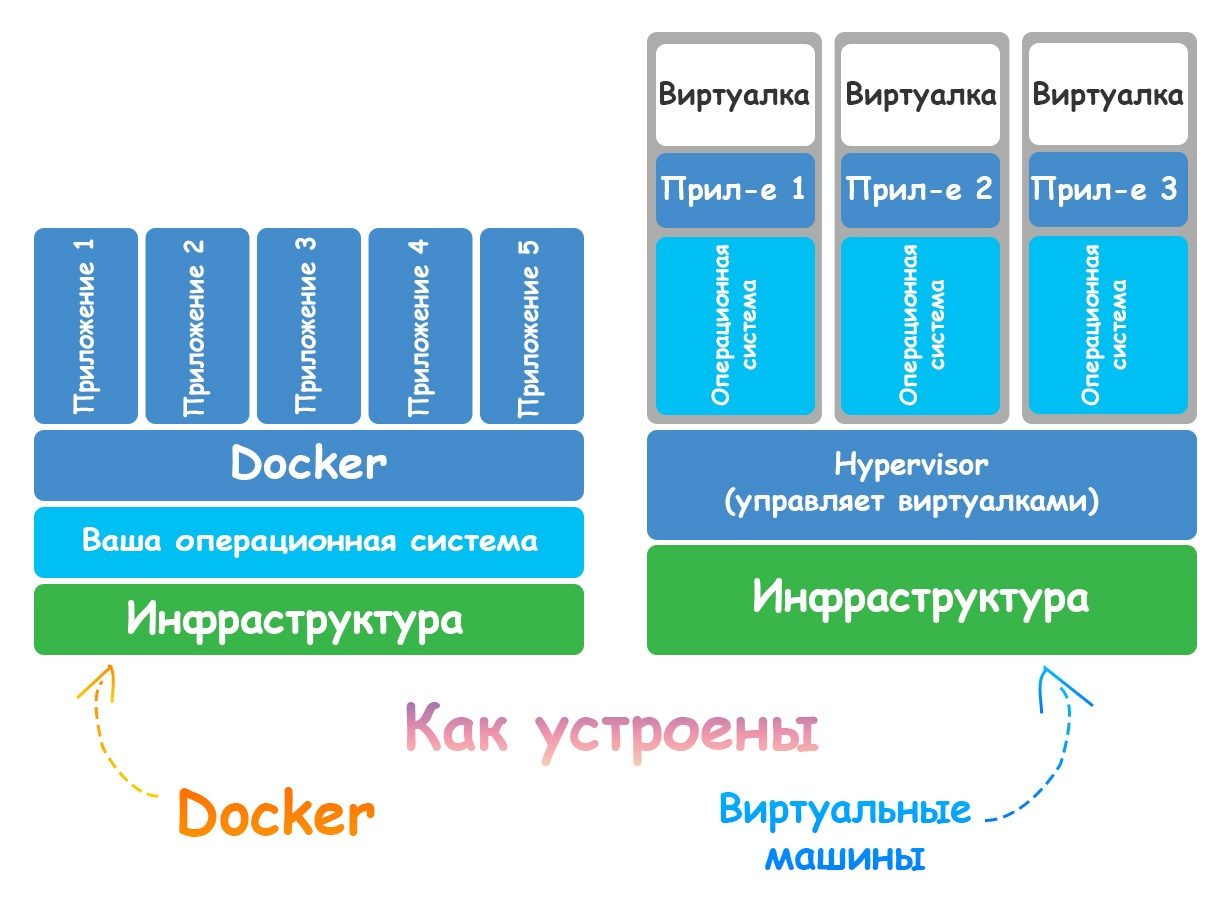


Рисунок [todo вставить номер рисунка] – Различие между Докером и традиционными системами виртуализации

Таким образом, под задачи, решаемые в рамках данной работы. Прекрасно подходит докер. Для того, что бы не разрабатывать докер контейнер полностью с нуля, в качестве основы будет использоваться стандартный контейнер с wiredaurd, причем большая часть настроек будет оставлено по умолчанию. Те настройки, которые должно упомянуть:

SERVERURL, хранящий в себе глобальный адрес сервера. Именно по данному адресу к контейнеру будут подключаться удаленные устройства. В рамках тестирования используется адрес нутрии локальной сети, т.е. по сути данную структуру можно было организовать и без использования wireguard, однако он необходим для соединения с терминалами, разнесенными в удаленные локальные сети

PEERS – максимальное количеств одновременно подключенных терминалов. Виду реализации сокетов не может быть больше 65534 для каждого порта, однако в случае наличия у сервера-оболочки нескольких соединений с внешним интернетом данную проблему можно решить используя паттерн проектирования «Фасад». Впрочем, для большинства решаемых задач данное ограничение не является критическим. В рамках тестирования выставлен лимит в 10.

Ports – соответствие между портом физического и виртуального устройства. В общем случае, может не совпадать, что удобно с точки зрения безопасности, однако может вызывать определенные сложности с конфигурированием и поддержкой. В рамках тестирования используется 51820

Таким образом, выше была описана основа файла docker-compose,определяющего виртуализацию и использование средств удаленной локальной сети для построение VPN соединения.

## Разработка тестового стенда.

Ввиду сложностей с доступом к настоящим терминалам, являющихся сертифицируемым и сложным дорогим устройством, было принято решение разработать приложение для тестирования подключения. Так как терминал бесконтактной оплаты имеет в своем программном обеспечении операционную систему, схожую с операционной системой Android, а также сходен по форм-фактору с большинством мобильных телефонов, было принято решение разрабатывать тестировочное приложение таким образом, чтобы его можно было запустить в том числе и на мобильных устройствах. Иными словами, для тестирования было разработано мобильное приложение.

Так как тестирование крайне важный процесс, а довольно большую часть рынка мобильных устройств удерживают аппараты на платформе IOS, было принято решение сделать приложение кроссплатформенным, для чего был выбран фреймворк для кроссплатформенной мобильной разработки ReactNative. Его выбор был обусловлен тем, что у данного фреймворка практически отсутствуют конкуренты – на сегодняшний день существует только 2 фреймворка для кроссплатформенной разработки, это ReactNative и Flutter. Каждый из них имеет свои преимущества и недостатки, однако ключевое различие между ними это используемый язык. Дело в том, что Flutter для написания кода использует свой специальный язык Durt, в то время как ReactNative позволяет писать на слегка видоизмененном JavaScript. Ввиду того, что весь остальной проект за исключением непосредственно ядра бэкенда написаны на JAvaScript, для единообразия кодовой базы было принято решение писать на ReactNative. Кроме того, что данный язык по сути является монополистом в области (учитывая невозможность использования Flutter по причинам . описанным выше), он еще и обладает рядом существенных преимуществ, таких как максимальная близость к нативным приложениям, по сравнению с подходом, предоставляемым программными решениями такими как Cordova, Ionic или Titanium, подробная и хорошо написанная документация, а также – возможность тестирования приложений непосредственно на этапе разработки.



Рисунок [todo вставить номер рисунка] – сравнение Flutter и ReactNative

Без сборки полноценных разматываемых сборок, посредством утилиты Expo, которая позволяет редактировать приложение практически в режиме реального времени, отслеживая изменения на экране мобильного устройства через несколько секунд после того, как они были внесены.

На текущий момент мобильное устройство тестирующее серверную систему имеет весьма ограниченный функционал – оно способно генерировать параметры терминала при запуске и подключаться к открываемому сокету для получения уведомлений от сервера. При этом в случае получения уведомления от сервера на экране устройства отображается всплывающее окно с информацией, характеризующей данное сообщение. Ввиду того, что терминал бесконтактной оплаты хоть и близок к мобильному устройству, но не в точности повторяет его, часть функционала протестировать не получится, например – в телефоне нет возможности без прав суперпользователя его перезагрузить, в то время как для терминала такой функционал может оказаться критичным. Тем не менее, функционал тестирующего приложения по мере развития программного продукта может и будет расширяться, либо путем добавления отдельных экранов, служащих для тестирования тех или иных возможностей сервера, либо путем добавления новых индикационных элементов на текущий.Стоит понимать, что тестовое приложение создается исключительно для тестирования разработчиками, а также. Вероятно, лицами, занимающимися тестированием проекта, поэтому для него не критично наличие хорошо проработанного пользовательского интерфейса. Примеры интерфейса, а также способ обращения к нему приведен ниже.



Рисунок [todo вставить номер рисунка] – главный экран тестировочного приложения

На рисунке можно видеть, что в общем случае для отправки запроса терминалу нет необходимости даже в специализированном приложении – достаточно обычного GET запроса, который может быть отправлен из браузера. При этом для решения проблем безопасности к данному запросу может быть без каких-либо сложностей дописан алгоритм авторизации по токену или паролю.

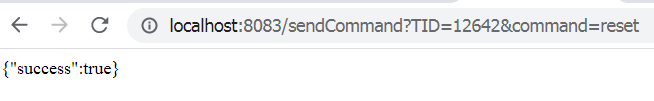


Рисунок [todo вставить номер рисунка] – Пример запроса из браузера   
При этом даже в текущей, базовой реализации такой запрос будет корректно обработан терминалом, что можно видеть исходя из рисунка ниже. Само собой, на данный момент в качестве обработки служит вывод пользовательских уведомлений, однако стоит понимать, что на момент выгрузки кода на терминал данный функционал будет расширен, в том числе методами, проприетарными для терминала.



Рисунок [todo вставить номер рисунка] – Пример эмуляции ответа на запрос

Текст кода программы предоставлен в репозитории Github. Ссылки на Github приведены в приложении В.

## Доработка пользовательского интерфейса

В ходе работы над тем объемом кода, который был передан для доработки ключевое внимание было уделено доработке клиентского части программного продукта. Для серверной части были дописаны 2 новых модуля, а также переработана логика подключения терминалов к серверу, однако изменения фронтенда были намного существеннее. В частности, на момент начала написания данной курсовой работы, вебстраница, служащая для мониторинга и управления терминами не поддерживала мобильные устройства. При попытке отобразить ее на телефоне или планшете отображалась стандартная верстка для настольных компьютеров, что является крайне нежелательной практикой в современной веб-разработке, так как в настоящее время зачастую сайты просматриваются именно с переносных устройств, и их доля приближается к 70%

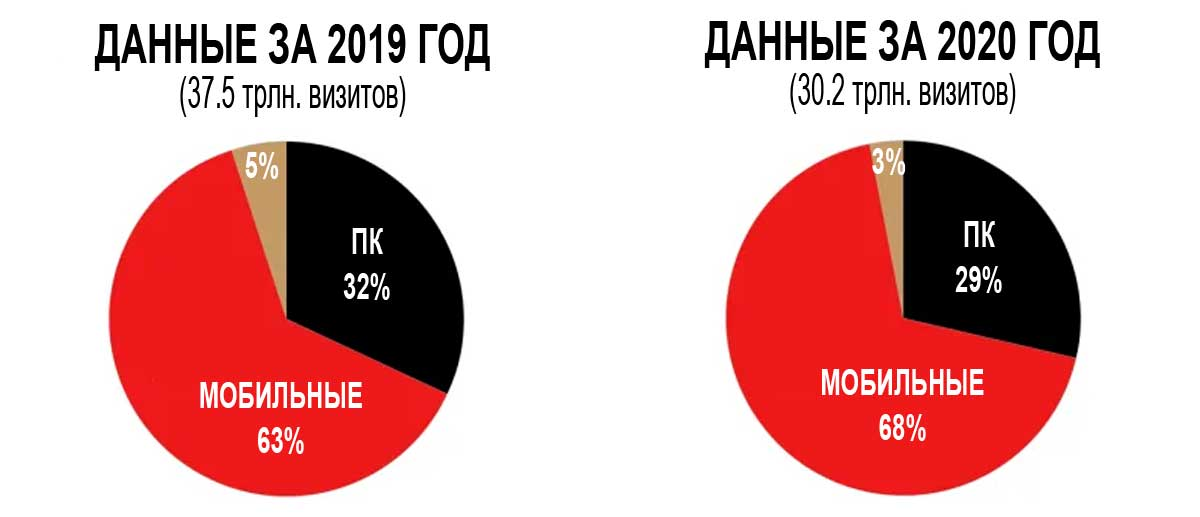


Рисунок [todo вставить номер рисунка] – соотношение между посещениями сайтов с персональных компьютеров и с мобильных телефонов

Таким образом, дизайн, который увидит две трети посетителей сайта в корне изменился – для них скрыто традиционное для сайтов из начала века боковое навигационное меню, которое появляется в адаптированном виде и только по нажатию специальной кнопки. Иначе говоря, был проведен редизайн всего клиентского приложения для пользователей мобильных устройств.   
 Другим крупным исправлением клиентской части приложения было добавление полной локализации. Теперь приложение само определяет тот язык, который предпочтителен для пользователя, и показывает ему все элементы с подписями на соответствующем языке. На данный момент поддерживаются только русский и английский, однако данный функционал построен таким образом, что для добавления новой языковой модели не нужно менять что-либо в логике – достаточно лишь расширить JSON файл, содержащий переводы каждого из компонента, так что для этого в общем случает даже не обязательно привлекать программиста. На данный момент язык определяется автоматически, путем исследования настроек браузера пользователя, однако возможно добавление функционала и для ручного переключения языковых предпочтений.

Еще одним нововведением является внедрение расширенной системы логгирования. Теперь при совершении какого-либо действия, в свеженаписанный микросервис логгирования отправляется запрос с требованием занести информацию о события в базу данных. Более подробно данный механизм описан в начале работы.

Текст кода программы предоставлен в репозитории Github. Ссылки на Github приведены в приложении В.

# Заключение

В рамках данной курсовой работы была существенно доработана система, служащая для мониторинга и конфигурирования банковских терминалов для приема бесконтактных платежей. Было разработано несколько программных модулей на серверной части приложения, существенно модернизирована клиентская часть приложения, а также разработан комплекс мер для тестирования и развертывания системы. Помимо вышесказанного, было произведено исследование предметной области, выявлены преимущества и недостатки используемых инструментов, а также проведен анализ сходных технических решений.

# Приложение А

Полный текст файла конфигурации со стороны сервера

|  |
| --- |
| version: "2.1" |
|  | services: |
|  | wireguard: |
|  | image: ghcr.io/linuxserver/wireguard |
|  | container\_name: WireGuard |
|  | cap\_add: |
|  | - NET\_ADMIN |
|  | - SYS\_MODULE |
|  | environment: |
|  | - PUID=1000 |
|  | - PGID=1000 |
|  | - TZ=Europe/London |
|  | - SERVERURL= 91.207.170.27 #optional #(глобальный адрес сервера) |
|  | - SERVERPORT=51820 #optional |
|  | - PEERS=1 #optional (количество хостов) |
|  | - PEERDNS=auto #optional (не нужно, DNS не испоьзуется) |
|  | - INTERNAL\_SUBNET=10.13.13.0 #optional |
|  | - ALLOWEDIPS=0.0.0.0/0 #optional |
|  | volumes: |
|  | - /path/to/appdata/config:/config |
|  | - /lib/modules:/lib/modules |
|  | ports: |
|  | - 51820:51820/udp |
|  | sysctls: |
|  | - net.ipv4.conf.all.src\_valid\_mark=1 |
|  | restart: unless-stopped |

# Приложение Б

Перечень команд к сервису сокетов с описаниями

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Команда | Служебное название команды | Текстовое описание | Используемый URL | Перечень передаваемых значений |
| Авторизация | Login | Команда первичной авторизации. Высылается перед началом работы с терминалом | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=login&pass=12345 | Pass – пароль доступа |
| Выход из учетной записи | Logout | Команда завершения работы в авторизованном режиме. | [http://127.0.0.1:8083/sendcommand?tid=0000 &command=logout](http://127.0.0.1:8083/sendcommang?tid=0000&command=logout) |  |
| Смена пароля | Change | Команда смены пароля. Принимает на вход старый пароль и новый пароль, в случае корректности – старый пароль заменяется новым | [http://127.0.0.1:8083/sendcommand?tid=0000 &command=change&pass=12345&newPass=54321](http://127.0.0.1:8083/sendcommang?tid=0000&command=change&pass=12345&newPass=54321) | Pass – старый пароль доступа, newPass – новый пароль доступа |
| Запрос загрузки обновления | Update | Вынуждает терминал проверить актуальность версии программного обеспечения и – в случае его устаревания – обновить его. | [http://127.0.0.1:8083/sendcommand](http://127.0.0.1:8083/sendcommang) ?tid=0000&command=update |  |
| Загрузка ключей-хозяев | Lmk | Получает чеки по использованию мастер-ключей | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=lmk |  |
| Загрузка рабочих ключей | Lwk | Получает чеки по использованию рабочих-ключей | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=lwk |  |
| Проверка связи к серверу | Test | Проверка наличия подключения между терминалом и сторонним сервером. | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=test |  |
| Получения информации о терминале | Param | Получение полной информации о терминале. Передает все конфигурационные и системные данные, а также, если возможно, данные о текущей геолокации | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=param&pass=12345 |  |
| Очистка журнала | Clear | Удаляет все операции из локального журнала транзакций. | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=clear |  |
| Сброс пароля | reset | Сбрасывает пароль на пароль по умолчанию. Требует пароля. | [http://127.0.0.1:8083/sendcommand?tid=0000](http://127.0.0.1:8083/sendcommang?tid=0000) &command=reset | Pass – пароль доступа |

# Приложение В

Перечень команд к сервису сокетов с описаниями

|  |  |
| --- | --- |
| Название службы | Ссылка на Github |
| Файл конфигурации для Docker | https://github.com/Toxa-p07a1330/diplom/blob/master/docker-compose.yml |
| Файлы исходного кода для тестового приложения | https://github.com/Toxa-p07a1330/diplom/tree/master/kds2/mobile |
| Файлы исходного кода клиентской части системы | https://github.com/Toxa-p07a1330/diplom/tree/master/kds2/front |
| Файлы исходного кода микросервиса логгирования | https://github.com/Toxa-p07a1330/diplom/tree/master/kds2/nodeJS/logServer |
| Файлы исходного кода микросервиса обеспечения соединения с терминалом | https://github.com/Toxa-p07a1330/diplom/tree/master/kds2/nodeJS/socketServer |
| Файлы исходного кода основного серверного приложения для работы с базой данных | https://github.com/Toxa-p07a1330/diplom/tree/master/kds2/src |