|  |  |
| --- | --- |
| Gerb-BMSTU_01 | Министерство науки и высшего образования Российской Федерации  Федеральное государственное бюджетное образовательное учреждение  высшего образования  «Московский государственный технический университет  имени Н.Э. Баумана  (национальный исследовательский университет)»  (МГТУ им. Н.Э. Баумана) |
| ФАКУЛЬТЕТ «Машиностроительные технологии»  КАФЕДРА «Литейные технологии»  РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА  К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ  НА ТЕМУ:  «Разработка системы информационной по осуществлению управления терминалами безналичных платежей…»  Студент группы ИУ3-81Б А. Т. Марчук  (Подпись, дата)  Руководитель ВКР И. М. Сидякин  (Подпись, дата)  Нормоконтролер С. С. Семушкин  (Подпись, дата) | |
| 2022 г. | |

# РЕФЕРАТ

Расчетно-пояснительная записка 36 с., 7 рис., 5 табл., 7 источников.

ПРОКАЛОЧНЫЕ ПЕЧИ, ЮВЕЛИРНЫЕ УКРАШЕНИЯ, ЛИТЬЕ ПО ВЫПЛАВЛЯЕМЫМ МОДЕЛЯМ, ФУТЕРОВКА

Объектом разработки является прокалочная печь для ювелирного литья.

Цель работы — модернизация прокалочной печи для уменьшения расхода энергии, затрачиваемой на нагрев печи.

Поставленная цель достигается за счет применения улучшенной футеровки печи и дверцы загрузочного окна, улучшения герметизации рабочего пространства печи за счет наклонной дверцы, замены нихромовых нагревательных элементов на силитовые стержни с целью создания стабильного температурного поля по сечению печи.

# СОДЕРЖАНИЕ

[РЕФЕРАТ 2](#_Toc100768570)

[СОДЕРЖАНИЕ 3](#_Toc100768571)

[ВВЕДЕНИЕ 4](#_Toc100768572)

[1. Исследовательская часть 7](#_Toc100768573)

[1.1. Глоссарий предметной области 7](#_Toc100768574)

[1.2. Особенности безналичных платежей 10](#_Toc100768575)

[1.3. Механизмы осуществления платежей 11](#_Toc100768576)

[1.4. Общая структура POS системы 12](#_Toc100768577)

[1.5. Анализ существующих решений в данной предметной области 13](#_Toc100768578)

[1.6. Основные проблемы с безопасностью и способы их решения 15](#_Toc100768579)

[1.7. Базовые принципы туннелирование и обоснование его необходимости 15](#_Toc100768580)

[2. Конструкторская часть 19](#_Toc100768581)

[2.1. Выбор и обоснование используемых технических средств 19](#_Toc100768582)

[2.2. Разработка базовой структуры системы 20](#_Toc100768583)

[2.3. Преимущества и недостатки монолитного и микросервинсого подхода 21](#_Toc100768584)

[2.4. Реализуемая архитектура системы 23](#_Toc100768585)

[2.5. Структураная организация проекта 25](#_Toc100768586)

[2.6. Выбор системы контроля версий 30](#_Toc100768587)

[3. Технологическая часть 37](#_Toc100768588)

[3.1. Система логгирования событий 37](#_Toc100768589)

[3.2. Микросервис сокетов 39](#_Toc100768590)

[3.3. Настройка VPN 42](#_Toc100768591)

[3.4. Разработка конфигурации для сервисов виртуализации 47](#_Toc100768592)

[3.5. Разработка тестового стенда 50](#_Toc100768593)

[3.6. Доработка пользовательского интерфейса 54](#_Toc100768594)

[ЗАКЛЮЧЕНИЕ 57](#_Toc100768595)

[Приложение А 61](#_Toc100768596)

[Приложение Б 62](#_Toc100768597)

[Приложение В 65](#_Toc100768598)

# 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.

Кроме того, учитывая текущую ситуацию с зарубежным программным обеспечением максимально остро встает вопрос разработки системы управления банковскими процессам, в частности – терминалами – исключительно силами внутреннего рынка, минифицировав внешние зависимости, для наличия возможности осуществлять финансовые операции, а также менеджмент терминалов при любых обстоятельствах, включая отсутствие доступа глобальный интернет и невозможность установки внешних эквайеринговых систем.

Нельзя не признать, что разработка данной системы совсем с нуля на данный момент не представляется возможной – по крайней мере, за сколь-либо приемлемое количество человекочасов. Даже реализуя с нуля приложение, в нем так или иначе будут использоваться разнообразные готовые программыне решения, например – фреймворки или библиотеки. По большому счету – с этим ничего нельзя сделать, единственная альтернатива – писать приложение полностью вручную в условиях реальной жизни выглядит утопически.

Однако стоит заметить, что даже в случае прекращения поддержки используемых библиотек ключевая логика приложения, которая была реализована в рамках дано работы сохранится, и для того, что бы заставить систему управления банковскими терминалами снова работать достаточно будет заменить бибилотеки на доступные или написать самостоятельно фреймворк с совместимым программным интерфейсом.

Вне зависимости от внешних обстоятельств, бесконтактные платежи уже крайне плотно вошли в нашу повседневную жизнь и большая часть из нас пользуется ими ежедневно. Развитие систем бесконтактной оплаты требует соразмерного развития прилегающей инфраструктуры – терминалов, банкоматов и так далее. И чем развитее и разветвленнее становится данная инфраструктура – тем сложнее становится ею управлять, и тем – как ни странно – важнее становится ее отказоустойчивость. Если какие-то лет 10 назад бещзналичный расчет воспринимался как дополнительная опция и большая часть проводилась при использовании бумажных денег, то в настоящее время сформировался целый пласт людей, кто бумажные деньги практически не использует – и в случае неполадок терминала бесконтактных платежей мгновенно становится неплатежеспособен. Хорошо это или плохо – вопрос открытый, который, вдобавок, выходит за рамки рассмотрения данной работы, однако так или иначе он подводит к мысли о том, что разработка системы управления терминалами банковских платежей является по настоящему актуальной и полезной задачей, которую надлежит решить в максимально короткой перспективе, и которая может оказать в дальней перспективе существенное влияние на облик финансовых операций.

## Исследовательская часть

### Глоссарий предметной области

Введем несколько определений и аббревиатур для дальнейшего использования.

EMV – Стандарт взаимодействия типовых карт в целях проведения оплаты по карте

EMV приложение – Микропрограмма в совокупности с блоком параметров на карте, относящейся к международной системе оплаты EMV

ПИН-пад – Электронно-цифровое устройство, использемое для введения PIN-кода для подтверждения доступаPIN код – Секретный цифровой код, который вводит клиент, в целях подтверждения того факта, что он имеет полномочия по использованию карты

Терминал – Аппаратный и программный комплекс средств, используемый в целях регистрации приобретения товара или услуги а также для упрощения приобретения товаров или услугКод действия терминалов (TAS) – Действия программного обеспечения терминалов в случае возникновении какой-либо конкретной ситуации (к примеру, отказ от операции, проведение операции с использованием сетевых банковских ресурсов или выполнение действия по умолчанию, например, перезагрузки)

Коммуникационный скрипт – малая по объему часть прогаммного обеспечения в форме последовательности команд, которая определяет сценарий установки соединения между терминалом и сервером аутентификации или TMS

Держатель / Клиент– Физическое лицо или же представитель юр.лица, на паспортные даные которого ввиду заключенного с Эмитентом контракта или на ПД которого по воле лица, заключителя договора с Эмитентом, был осуществлен выпуск Карты.

Магнитная полоса - Покрытая ферромагнетиком полоса на карте, которая содержит содержащаяидентификационную информацию как о карте, так и о лице, являющемся ее держателем.

Операция - Банковская расчетная операция, которая осуществляется при использованием Карты при условии обязательной аутентификации в соответствии с текущим законодательством России, НПА Банка РФ и контрактом с Эмитентом, которая проводится в Торговой точке в соответствии с требованием Держателя карты.

Операция возврата - Банковская расчетная операция, которая совершается при использовании Карты в случае возникновении необходимости осуществления возврата Держателю денежных средств, которые были списаны с его счета по причине операции, совершенной ранее.

Операция отмены - Выполняемая до формирования эл.журнала Процедура отмены авторизационного кода.

Электронный журнал - Возможно единичная совокупность документов, хранимых и обрабатываемых в электронной форме, которая была сформирована за некий определенный период времени в результате совершении Операций, в том числе с использованием POS-терминала, но не исключительно с ним.

Программное обеспечение (ПО) -совокупность программных решений POS-терминала, служащих для его работы.

Профиль - объект системы, служащий для хранения параметров групп параметров, которые определяют алгоритм работы терминала в ходе выполнения административных операций или операции с картой..

Реквизиты Карты - срок обслуживания Карты, Номер Карты , код безопасности CVV2/CVC2. Для рзличных карт перечень реквизитов может несколько отличаться

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

Сервисный код коммерсанта – Четырециферный код, обозначающий профиль деятельности, коммерсанта.

Скрипт эмитента - Команда или совокупность команд, которая передается от эмитента к терминалу для выполнения микропроцессорной картой полученной команд или системы команд.

Способ верификации держателя карты - Способ подтверждения, что человек, который предъявляет карту, уполномочен на ее использование. Примером может служить введение PIN-кода либо же подпись  
Справочник - Объект системы, используемый для хранения основных  
значений, шаблонов, терминов и алгоритмов, которыми оперирует система.  
Транзакция - Группа последовательных действий, представяющая собой логическую единицу обработки данными. Транзакцию можно выполнить либо полностью успешно, при соблюдении целостности данных и вне зависимости от других параллельных транзакций, либо не будет выполнена совсем и в тком случае она не должна производить какого-либо эффекта. Транзакции подлежат обработке транзакционными системами, во вреся работы которых происходит создание 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, которая осуществляет весь спектр операций в вопросе взаимодействию с точками обслуживания платежных карт, состоящая из терминалов в сети ТСС и банкоматов.  
  
Эмитент - Организация, выпустившая (эмитировавшая) карточки для развития и  
финансирования своей деятельности  
AID - Application Identification Number (Идентификационный номер приложения)  
AVN -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

### Особенности безналичных платежей

Как и упоминалось выше, исторически первыми способами платежей были расчеты с помощью наличных стредств. Покупатель отдавал продавцу деньги, а взамен получал какой-либо товар или услугу.

Рассмотрим плюсы и минусы данной формы расчетов с позиции продавца и покупателя, а кроме того с точки зрения глобальной государственной экономики.

Таблица 1 – анализ наличных и безналичных расчетов с точки зрения обывателя.

|  |  |
| --- | --- |
| Анализ наличных расчетов с точки зрения продавца и покупателя | |
| Плюсы | Минусы |
| Предоставение как покупателю, так и продавцу полную свободу и независимость от внешнего влияния. Главное условие – надежная защита дензнаков от подделки и достаточное количество их в обороте.  В значительной степени поддерживаю анонимность платежей – без переписывания номеров купюр отследить передачу денег практически невозможно. | Неудобство, а и зачастую и невозможность совершения передачи денежных средств без личного физического взаимодействия участников денежной операции. Необходимо обеспечивать условия для безопасного хранения денег. Риск получения поддельных денежных знаков. Сложность передачи денег на большие расстояния. |

Таблица 2 – анализ наличных и безналичных расчетов со стороны государственной экономики

|  |  |
| --- | --- |
| Анализ наличных расчетов с точки зрения государства | |
| Плюсы | Минусы |
| Сложившийся в силу исторических причин формат расплаты, к которому привыкло население. | В случае расчетов в наличных деньги хранятся у продавцов и по сути выводятся из оборота, до тех пор пока продавцом не будет совершена на них какая-либо покупка. |
| Государство имеет инфраструктурные траты на производство денег, их логистику, хранение и утилизацию. |
| Наличные расчеты в малой степени подконтрольны фискальным органам, в связи с чем способствуют развитию нелегальной экономики и провоцируют уход от уплаты налогов. |

Исходя из этого,  видно, что наличный расчет для государства нежелателен, которое оно запретило, в случае если это не вызвало бы резкой негативной реакции населения. Ввиду того, что полностью запретить наличный расчет в ближайшее время не представляется возможным, то применяют целый ряд ограничительных мер.

В России законодательно (ГК РФ ст. 861, Указание Банка России 3073-У от 07.10.2013) устанавливает, что только физические лица и только в личных целях могут осуществолять использование наличных денежных средств без ограничений, для юридических лиц же применение наличности подвержено лимитированию.

Безналичные платежи, в числе прочего, подразумевают присутствие не только продавца и покупателя, но и некой третьей доверенной стороны, выполняющей функцию посредника, который по воле сторон осуществляет расчеты между ними. При этом необходимым условием является надежность посредника – ему должны безусловно доверять как продавец, так и покупатель.

Криптовалюты, позволяют осуществлять платежи напрямую, без посредников, но на данный момент их статус законодательно не определен. Кроме того, данная финансовая область не относится к теме данной работы.

### Механизмы осуществления платежей

Проанализируем, как производится безналичная оплата между отправителем и получателем (далее для их обозначения будет использоваться понятие клиенты), получающих обслуживание в одном банке.

Плательщик может добровольно приказать банку осуществить платеж в пользу получателя в счет средств, хранящихся на своем счету – платежи по платежным поручениям.

Получатель вправе затребовать у банка, держащим счет плательщика, осуществить платеж в свой адрес в случае наличия договоренности с плательщиком или в иных случаях, зафиксированных в законодательстве. Платеж может быть совершен с подтверждения плательщика – в таком случае это расчет по платежным требованиям, или же в без акцептировании, например расчет по т.н. инкассовым поручениям.

Плательщик и получатель вправе договориться о том, совершении банком платежа в адрес получателя в случае предъявления в банк-эквайер последним заранее определенных документов, подтверждающих осуществление совершения сделки.

Кроме того, возможны другие формы, описанные в п.1.1 Положения Банка РФ от 19.06.2012 N 383-П «О правилах перевода денежных средств».

Для проведения межбанковских переводов банки должны между собой установить корреспондентские отношения. Суть данных отношений в том, что Банк 2, становится клиентом Банка 1 и затем осуществляет открытие в нем специального счета, который называется корреспондентским счетом. В результате открытия корреспондентского счета Банк 2 должен внести на него определенное количество денег, своего рода денежный буфер, в пределах которого клиенты Банка 2 будут способны высылать платежи клиентам Банка 1.

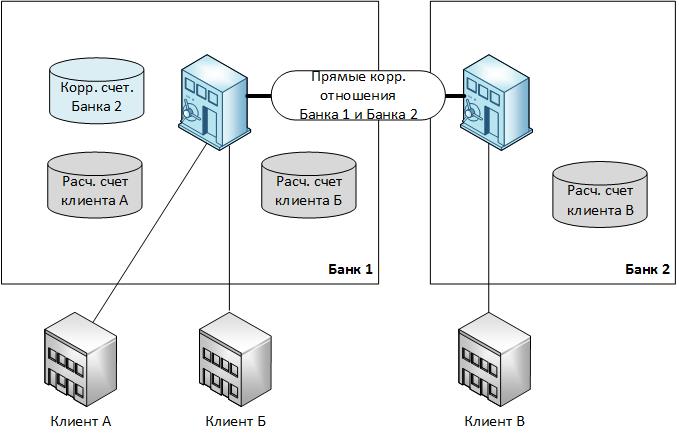


Рисунок 1 – схема межбанковского перевода

В рассматриваемом случае не используются терминалы бесконтактной оплаты, но по большому счету они только добавляют лишний уровень абстракции для конечных клиентов, в результате они не сами осуществляют приказ банку, но общаются с банком посредством терминала.

### Общая структура POS системы

Современные POS-системы представляют из собя совокупность аппаратных и программных решений, которые позволяют осуществлять платежные операции и упрощающие ежедневные бизнес-процессы. Если речь заходит о POS-ах, как правило имеют в виду терминалы оплаты, кассовые аппараты и другие заурядные средства обслуживания торговых магазинов. Однако стоит отметить, что архитектура POS не ограничена исключительно данными компонентами. Пример осуществления процедуры транзакции изображен на рисунке 1. Стоит отметить, что на рисунке 1 приведена упрощенная схема, служащая для демонстрации участников взаимодействия, в то время как полная схема, отображающая данную процедуру более подробно, с обозначенными вариативностями и отображением систем подтвержений приведена в приложении [todo сосласться на приложение]



Рисунок 2 – пример полной процедуры транзакции

Сначала держатель прикладывает карту к считывателю. Данные карты высылаются в терминал, откуда поступают в POS-систему. Затем, POS-система обращается к сервису обеспечения платежей, который в свою очередь, в зависимости от разновидности кредитной карты, посылает запрос в банк для осуществления процедуры авторизации перевода. В этот момент держателю предоставляется возможность ввода PIN-код того, чтобы подтвердить транзакцию. В случае, если все прошло успешно, авторизационный код возвращается из банковской локальной сети в PSP и отдается 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

### Основные проблемы с безопасностью и способы их решения

Говоря о банковских операциях нельзя не затронуть вопрос безопасности, ведь именно в данной сфере он стоит максимально остро. Для обеспечения безопасного соединения испольуются множество средств, обеспечения безопасного соединения.

Стоит отметить, что решение о выдаче денег банкоматом никогда не производится локально – в целях безопасности, для обработки каждой из транзакций он обращается к расположенному в банке серверу. Подключение к этому серверу,, который называется процессинговым центром, осуществляется по проводным или беспроводным каналам связи (например, через оптоволокно). Критическим важно, чтобы соединение было защищенным от перехвата данных, поэтому чаще всего используются программные или аппаратные VPN-клиенты.

Чаще всего, обмен данными с процессинговым центром осуществляется по протоколам NDC или DDC, однако некоторые банки используют и решения, реализованные самостоятельно. Помимо процессингового центра терминал также имеет соединение с локальной внутренней сетью банка, которая используется для подключения в целях удаленного администрирования, а также с сервером обновления ПО.

### Базовые принципы туннелирование и обоснование его необходимости

Если речь заходит о безопасности, не является хорошей практикой разрабатывать полностью самодельные решения – зачастую они оказываются не самыми надежными и оптимальными. В связи с этим, для обеспечения безопасности соединения между терминалом и сервером будет осуществляться туннелировапние канала связи посредством сервиса VPN, а для обеспечения защиты канала связи между клиентской частью сайта и сервером будет использоваться защищенная версия сетевого протокола взаимодействия - https. Для более полного раскрытия вопроса в данном разделе стоит проиллюстрировать как работает каждая из этих технологий.

Для начала, стоит обозначить, что VPN по расшифровке есть виртуальная частная сеть, со всеми проистекающими плюсами и минусами. Иными словами, компьютеры, расположенные в единой сети VPN логически находятся в единой локальной сети, даже если физически они расположены в абсолютно разных местах.

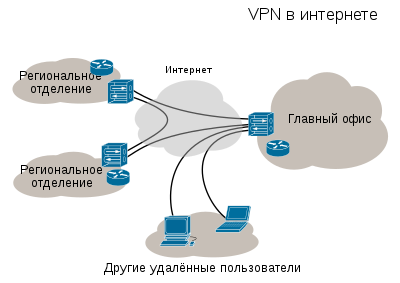
.

Рисунок 3– схема работы VPN

При этом стоит отметить, что даже несмотря на то, что трафик казалось бы проходит через глобальные сети, как минимум на уровне провайдера, ни редактировать его, ни даже менять провайдер не вправе. Для осуществления решения задачи шифрования служат защищенные протоколы. Сравнительная таблица протоколов VPN приведена в приложении 1

В рамках данной работы используется протокол шифрованного соединения WireGuard, который в полной степени удовлетворяет требованиям безопасности.

Что касается вопроса https соединения, его безопасность в настоящее время осуществляется с помощью протокола TLS, расширенной версии протокола SSL.

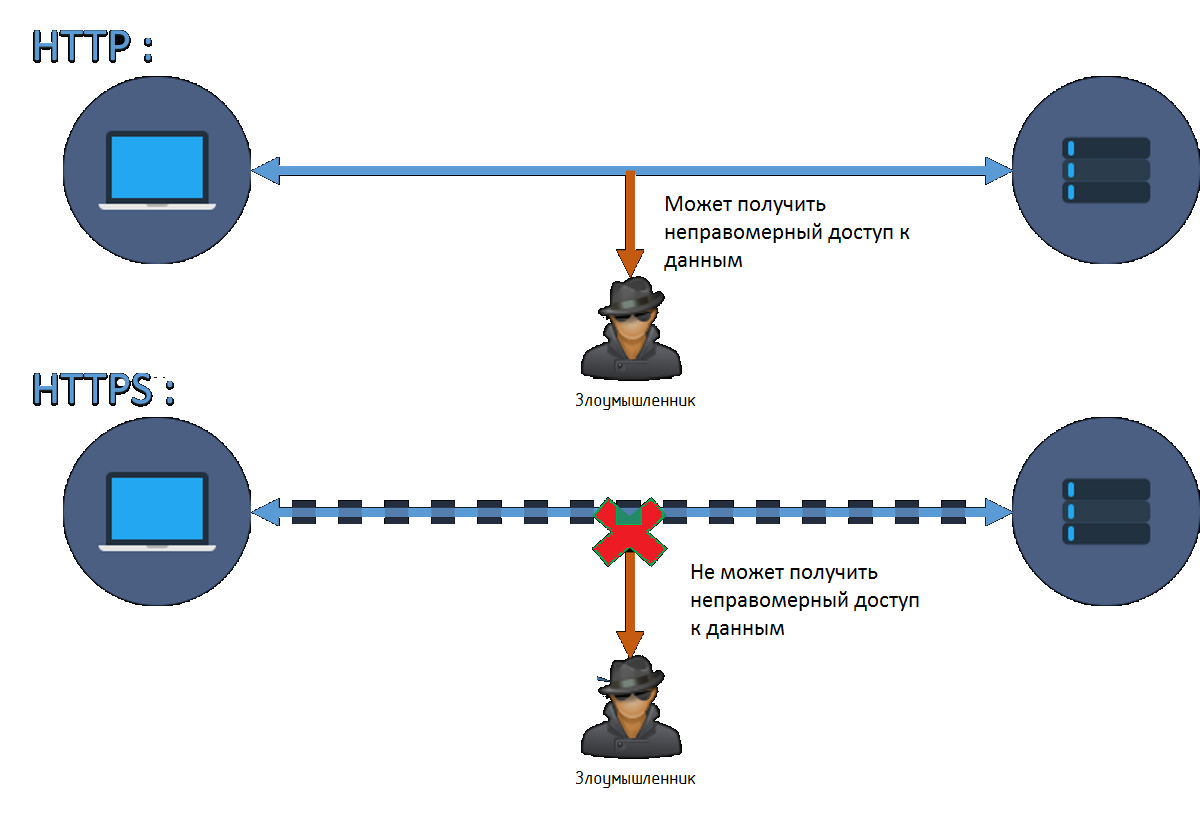


Рисунок 4– иллюстрация различия протоколов HTTPS и HTTP

TLS – является гибридной криптографической системой. Это означает, что в ней используется несколько различных криптографических подходов, которые будут рассмотрены далее:

1) Асимметричное шифрование (также называется криптосистема с открытым ключом) используется для генерации общего приватного ключа и аутентификации.

2) Симметричное шифрование, которое использует приватный ключ для последующего шифрования запросов и ответов.

Криптосистема с открытым ключом – это подвид криптографической системы, в которой у каждой из сторон есть и публичный, и приватный ключи, являющиеся связанными числами, например – взаимно простыми. Публичный ключ используется для кодирования текста сообщения в нечитаемую последовательность бит, в то время как приватный ключ напротив, используется для расшифрования в целях получения исходного прямого текста.

После того, как сообщение зашифровано с помощью публичного ключа, его можно расшифровать только с помощью соответствующего ему приватного ключа. Ключи не являются взаимозаменяемыми. Публичный ключ может публиковаться в свободном доступе без какого-либо риска для безопасности системы, однако приватный ключ не должен быть передан лицу, которое не имеет прав на расшифровку данных.

Одним из наиболее значимых преимуществ ассиметричного шифрования можно назвать то, что как получатель, так и отправитель, не осуществлявших обмена информацией до этого, способны установить безопасное соединение, даже обмениваясь сообщениями по незащищенному каналу связи.

Как клиент, так и сервер используют различные и секретные собственные приватные ключи и опубликованный публичный ключ для генерации общего секретного ключа для установленной сессии.

Таким образом, если кто-либо перехватит сообщения, находясь между клиентом и сервером, он не сможет узнать ни одного из ключей, ни тем более будет неспособен расшифровать перехваченное сообщение.

## Конструкторская часть

### Выбор и обоснование используемых технических средств

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. Ввиду невозможности использования реального тестового терминала на этапе тестированию будет написано кроссплатформенное приложение для смартфонов, которое позволит с достаточной степенью точности сымитировать его поведение. Для этого будет использоваться ReactNative. Он был выбран в связи с тем, что подходы при проектировании приложений на ReactNative во многом совпадают с подходами проектирования на ReactJS, а в связи с этим совпадают и преимущества. Кроме того, сравнивая ReactNative с другими фреймворками для разработки мобильных приложений нельзя не отметить, что в нем крайне удобно для разработчика реализован функционал выбора целевой платформы – IOS или Android, а также имеет место быть простая и надежная система сборки тестовых билдов – Expo, которая позволяет тестировать приложения без полноценной установки, что может быть полезно как в вопросе интеграционного тестирования внедряемых возможностей, так и для проведения массовых тестов нескольких приложений – для того, чтобы переключаться между объектами тестирования достаточно просто выбрать в лаунчере Expo Mobile

Рассматривая выбранный стек с точки зрения доступности, упомянутой в введении, нельзя не признать, что он не является полностью внутренним и значительная часть зависимостей, из тех, которые были перечислены выше, могут нуждаться в замене. Так, система управления базами данных MySQL, которая использовалась в проекте с момента начала его написания, начиная с 2010ого года принадлежит АО «Oracle», которая на момент написания данной работы приостановила лицензирование на территории РФ. Кроме того, для написания основного микросервиса серверной части приложения, использовался Spring Framework, который разработан VMWare Inc, которая, в свою очередь, пока что не прекратила свою деятельность, но публично заявила о том, что планирует это сделать.

Ряд сложностей также может возникнуть и с ReactJS, и с WireGuard, однако на момент написания никаких ограничений с их стороны не предвиделось.

В связи с этим, стоит упомянуть альтернативные платформы, которые могут быть использованы в случае, если с доступом к платформам, перечисленным выше, возникнут трудности.

Для реализации клиентской части сайта можно воспользоваться решением с открытым программным кодом JQuerry. В настоящее время он морально устарел, однако данное программное решение разрабатывается добровольцами на пожертвования, в связи с чем является некоммерческим проектом, которым может пользоваться каждый без ограничений, в рамках лицензии.

Для реализации серверной части сайта может использоваться платформа Parser, которая была разработана и поддерживается на локальном рынке. Система управления базами данных может быть тоже полностью заменена на российский Линтер, который хоть и имеет менее обширный функционал, чем крупные международные систему управления данными, тем не менее вполне достаточен для решения задач, поставленных в рамках данной работы.

### Разработка базовой структуры системы

При выполнении данной выпускной-квалификационной работы использовалось сразу 2 подхода – как монолитной, так и микросервисной разработки. Стоит отметить, что определенная часть презентуемого объема кода была унаследована из наработок кафедры по данному вопросу, которые писались достаточно давно. На момент написания базовой части системы понятие о разработке микросервисных систем еще не было в достаточной степени проработано, поэтому изначально система писалась в монолитной архитектуре. Однако на тот момент, когда данная система была отдана на доработку, парадигма монолитной архитектуры окончательно ушла в прошлое и ее заместила архитектура микросервисная.

### Преимущества и недостатки монолитного и микросервинсого подхода

Каждый из подходов, как монолитный, так и микросервисный имеет как свои плюсы, так и свои минусы. В частности, к плюсам монолитной архитектуры принято относить такие параметры, как Упрощенная разработка и развертывание сниженное количество так называемых сквозных проблем, более высокую производительностью. На каждом из них стоит остановится более подробно.

В частности, упрощенное разработка и развертывание означает, что все действия осуществляются на единственном каталоге, без отсутствия необходимости настройки большого количества повторяющихся или перекрестных зависимостей, а также без необходимости осуществления согласования версий между различными микросервисами. Более того, для развертывания обновленной системы достаточно обновить на сервере лишь одну директорию, что как правило выполняется даже автоматически, без непосредственного участия человека. Настроить автоматическое обновление миногосервисной системы тоже возможно, однако данная процедура на порядок более сложная и так или иначе требует ручного контроля, ввиду того, что в ней существенно больше потенциальных точек отказа. В то же самое время, в случе, если монолитную систему удалось развернуть на сервере и она корректно работает – можно быть уверенным, что в ней не будет дальнейших проблем.

Малое количество сквозных проблем имеет в своей основе отсуствие настройки отдельного логгирования или контроля производительности каждого из компонентов системы. Ввиду того, что в системе только один компонент, можно быть уверенным, что если доступен он – доступна и вся система, а также нет необходимости при добавлении нового функционала отдельно настраивать логгирование, права доступа и иные атрибуты – все эти данные уже указаны для основного сервера, поэтому их достаточно настроить только единожды.

Самый весомый аргумент в пользу монолитных приложений до недавнего веремени был то, что у них более высокая производительность. Действительно, микросервисная архитектура предполагает активное взаимодействие между различными компонетами посредством сети, что в свою очередь ведет к необходимости принятия воо внимание сетевых задержек. Однако нельзя не отметить, что во первых, зачастую различные компоненты находятся на одном физическом сервере, ввиду чего сетевые задержки при взаимодействии компонентов минимальны, кроме того. Во вторых нельзя не отметить. Что с каждым годом пропускная способность сети становится все больше и больше, так что даже если сервера расположены территориально в различных центрах данных, через них все равно можно пропускать данные с скоростью достаточной для того, что бы серверная задержка оставалась пренебрежимо малой. Таким образом, этот недостаток хоть и имеет место быть, с каждым годом становится все менее и менее весомым.

С другой стороны, нельзя не упомянуть, что микросервисная архитектура имеет и свои плюсы. К основным принято относить максимальную гибкость приложений, простоту внедрения новых технологий, а также четкость отдельных логических чстей системы.

Для того, чтобы раскрыть более подробно каждый пункт из перечисленных ниже, стоит сказать, что существенной проблемой монолитных приложений является тот факт, что в них нельзя смешивать различные технологии. В частности, если приложение написано на языке Java, оно должно быть написано на Java полностью. Это сразу же отсекает огромное количество возможностей, как минимум для систем, которые разрабатываются не одним разработчиком, а командой из нескольких разработчиков. В частности, в истории существовало немало примеров систем, в которых различные микросервисы писались на различных языках программирования, с совершенно разными подходами. По понятным причинам, если уж отбросить совсем экзотические случаи, по типу ассемблерных вставок в С++, при разработке монолитной системы такое невозможно. Кроме того, стоит отметить, что помимо разлчиного перечня технологий, могут разнится и строки выполнения работ – в случае, если микросервис не готов к сроку, его можно сделать имитационным, возвращая статические значения и не опасаясь, что эта обманка как-либо повлияет на остальные элементы системы.

Тот же самый аспект касается и вопроса о внедрении новых технологий. В случае необходимости внедрения новых технологий, микросервисная архитектура позволяет осуществить не революционный, а эволюционный контроль, изменяя на новый перечень технологий не все компоненты системы единовременно, а перенося их по одному – и продолжая миграцию тогда и только тогда, когда удалось добиться работоспособности нового компонента в старом окружении. Отчасти данный вопрос связан с тематикой тестирования, которая будет рассмотрена ниже, однако он заслуживает и отдельного упоминания – в современном мире актуальность технологий крайне быстро пропадает, и тот стек, который использовался несколько лет назад, на сегодняшний день может бысть устаревшим и неподдерживающимся. Кроме того, нельзя не отметить тот факт. Что прогресс не стоит на месте и регулярно появляются как программные, так и аппаратные средства, которые осуществляют функции, выполнявшиеся их предшественниками быстрее, дешевле или качественнее. Помимо всего прочего, нельзя не отметить, чт при использовании устаревших программных продуктов могут возникнуть проблемы с актуальным окружением или даже с безопасностью, потому как использование устаревших защитных протоколов сильно упрощает получение неавторизированного доступа к защищенным данным. Особенно остро этот вопрос встает если вспомнить о том, что речь идет о безопасности банковской системы, где безопасность – а соответственно и актуальность протоколов принято ставить во главу угла.

Ну и говоря о тестировании нельзя не отметить, что систему с микросервисной архитектурой проще тестировать, потому как можно утверждать, что если корректно работает каждое отдельное звено микросервисной архитектуры – система работает в целом правильно, а тестирование множества раздельных независимых сущностей по сути своей сильно проще и эффективнее, чем тестирование большой единой сущности, как минимум потому что все зависимости между сущностями легко отследить и они не могут ввиду тех или иных программных ошибок или недоработок взаимодействовать не в том режиме, в котором задумано.

### Реализуемая архитектура системы

В связи с этим, было принято решение перехода разрабатываемой системы на микросервисную архитектуру. Кроме того, по причинам, изложенным выше, произошла частичная смена языка разработки – для микросервисной разработки больше подходил NodeJS. В связи с этим, в разработанной системе есть центральный сервис, написанный на Java и работающий с основным функционалом системы, но помимо него есть 2 вспомиогательных микросервиса – микросервис логгирования и микросервис сокетов.

Микросервис логгирования обеспечивает запись в базу данных информации о каждой из выполняемых операций. Он практически не связан логически с микросервисом ядра, вдобавок микросервис ядра может работать корректно даже в случае, если по тем или иным причинам микросервис логгирования будет отключен.

Микросервис сокета является более критичным. Он обеспечивает возможность управления терминалами через веб-интерфейс. Например, основной вариант работы – при открытии списка терминалов можно просмотреть перечень команд, которые можно выполнть с терминалом и после этого, путем нажатия клавиши из интерфейса, команда посредством сокета будет отправлена на терминал или на устройство, его эмулирующее. Данный микросервис более критичен к своей работоспособности, ввиду того что в случае выхода его из строя изчезнет возможность как управлять терминалами, как и получать о них обновленную информацию, однако вынесен как отдельный сервис он был по другим причниам – в частности, он выполняет принципиально другой функционал, нежели микросервис ядра – если последний обеспечивает исключительно взаимодействие с базой данных, то микросервис сокета является воистину интерактивным элементом. Кроме того, нельзя не отметить, что технологии на микросервисе ядра и на микросервисе сокета кардинально различаются – в частности, микросервис ядра ничего кроме обращения к базе данных не осуществляет, а микросервис сокета – напротив, лишь рассылает управляющие команды терминалам и не имеет соединения с базой данных в принципе. Обобщенная схема гибридной архитектуры системы приведена на рисунке [todo номер рисунка]

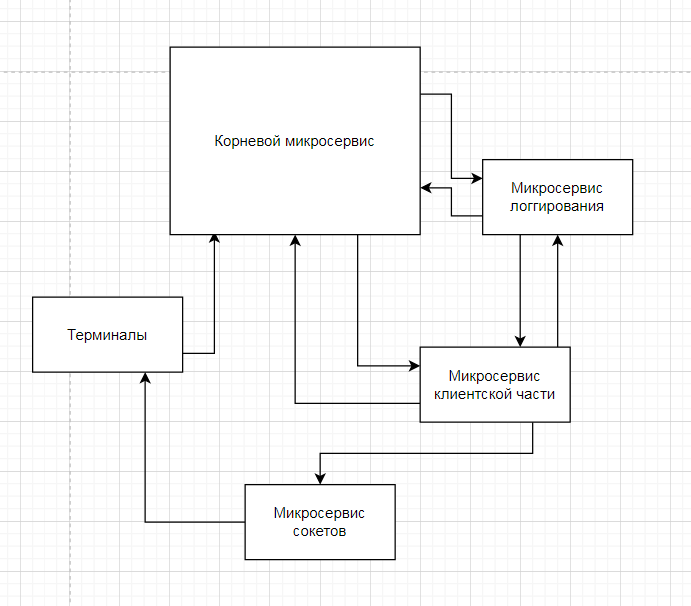


Рисунок 5 – базовая схема архитектуры системы

### Структураная организация проекта

Для хранения файлов исходного кода использовалась локальная директория в пользовательской операционной Window, к которойц был привязан репозиторий на хостинге системы контроля версий GitHub. В коне указанной директории расположено определенное множество сопросоводительных документов, таких как расчетно пояснительная записка или yml файл для конфигурирования контейнера Docker. Кроме этог, в отдельную папку Dumps вынесены файлы, описывающие структуру базы данных – на тот случай, если в процессе разрботки системы в структуру базы данных будут внесены необратимые изменения, эти файлы могут быть использованы для ее восстановления. В то же самое время стоит заметить, что необходимость в резервном копировании файлов исходных кодов программ не стоит столь остро, ввиду того, что для решения данной задачи хорош рабтает упомнянутая выше система контроля версий.

Основная директория, содержащая файлы исходного кода есть kds2. Это корневая папка всех файлов исходного кода проекта, однако изначально она была создана как хранилище монолитного сервера для серверной части приложения, поэтому в данной папке помимо сложной вложенной структуры из нескольких микросервисов есть и конфигурационные файлы для корневого микросервиса. В некотором смысле было бы корректно перенести хранимые в корне данной директории файлы в отдельную директорию, однако данное решение не повлияет на работоспособность сервиса или на удобство его доработок, с связи с чем было принято решение от него отказаться и выносить в поддиректрии ишь отдельные малые микросервисы. Структура каталга kds2 приведена на рисунке (todo ввести номер рисунка)

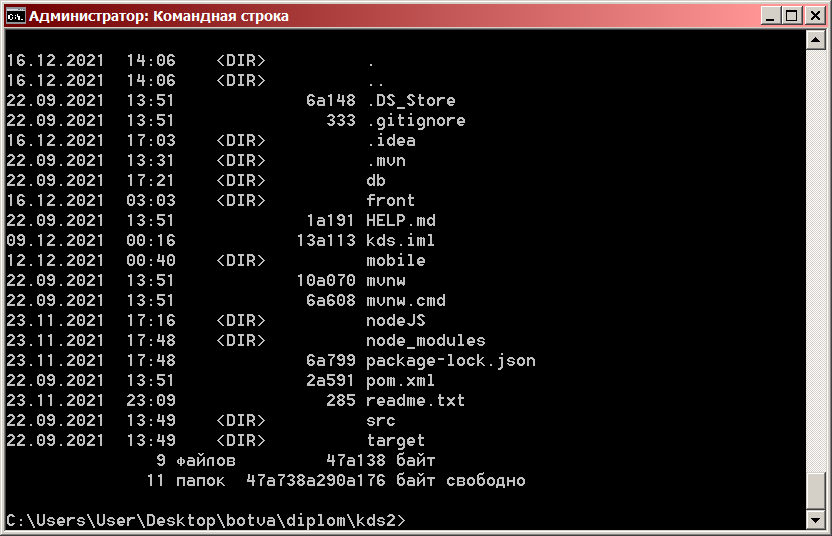


Рисунок 6– структура директории kds2

В частности, в отдельные папки вынесены такие разделы приложения как front, db, mobile и nodeJS, который в свою очередь разделен на подкаталоги logService и socketService. Каждый из вышеупомянутых каталогов содержит одноименный микросервис, однако не будет излишним внести некоторые дополнительные пояснения.

В частности, в каталоге mobile, если судить по названию, должно хранится клиентское мобильное приложение, чья разработка до настоящего момента в рамках данной выпускной квалификационной работы не упоминалась. Действительно, задачи разработки как такового мобильного приложения в рамках текущего приложения не ставилось, однако оно было разработано в целях тестирования и отладки основного функционала. Ввиду того, что целевая платформа, который предполагается управлять, есть довольно сложные в плане организации пользовательского доступа POS терминалы, было принято ращение разработки мобильного приложения, которое с некоторой конечной степенью достоверности может имитировать поведение терминала и при этом – является свободно копируемым и не заставляет испытывать трудности при распространении на сторонние мобильные устройства. Для того, что бы протестировать существенную часть взаимодействий между терминалом и сервером отныне нет нужды в подключении реального терминала – достаточно будет воспользоваться бюджетным мобильным телефоном с подключением к интернету и предустановленным приложением. Безусловно, нельзя утверждать, что подобная симуляция повторяет поведение терминала с абсолютной точностью, однако стоит заметить, что для решения большинства задач тестирования данного приложения хватает.

Каталог front, как можно понять из его названия, содержит в себе приложение. Которое должно быть отображено на клиентской стороне системы при обращении к ней посредством инструмента для просмотра веб-страниц. Внутри данного каталога тоже есть в значительной степени сложная и неоднородная структура, как. Собственно, и в большинстве иных микросервисов. Однако более подробно она будет рассмотрена в третьей главе, которая будет содержать подробное и детальное описание каждого из разработанных сервисов, а не краткий и поверхностный обзор системы в целом.

Директория NodeJS является родительской директорией для всех компонентов системы, которые написаны с использованием фреймворка для серверной разработки NodeJS. Стоит еще раз отметить, что сервис получился полиязычный, поэтому имеет смысл для различных наборов средств разработки делать как минимум различные директории. В более полном смысле для разных микросервисов принято выделять даже различные репозитории, но в силу ряда объективных причин, как то малые масштабы проекта, необходимость наличия единой точки доступа ко всем ресурсам и отсутствие необходимости ограничения доступа к системе третьим лицам.

Нельзя не упомянуть наличие некоторого количества служебных директорий, таких как target или src, однако они не представляют интереса, ввиду того, что не принимают участия непосредственно в процессе разработки, и, помимо всего прочего, являются автоматически сгенерированными с стандартным содержимым. В связи с вышесказанным, с перечнем служебных директорий можно ознакомиться на рисунке [todo ввести номер рисунка], но подробного описания их предназначения и содержания в данной работе не будет ввиду отсутствия необходимости.

Кроме того, стоит кратко упомянуть о используемой системе контроля версий, а также о том, почему было выбран именно такой подход.

Вопреки традиционным подходам, именно корневая директория была выбрана в качестве корня репозитория. таким образом. При каждой инициализации процедуры git add в локальный репозиторий будут записываться не только изменения исходного кода, но и изменения документации. С одной стороны, нельзя не признать, что это является неоспоримым преимуществом, ввиду того, что данный паттерн управления версиями позволяет отслеживать не только динамику изменения состояния файлов исходного кода, но и изменения приложенной документации, что, в свою очередь, гарантирует ее актуальность и соотнесенность с правками исходного кода, но и позволяет четко знать какие именно правки и в какой период были туда внесены. С другой стороны, стоит отметить, что в силу объективных причин, выгрузка сопровождающей документации в систему контроля версий имеет и свои недостатки, к которым принято относить необходимость выгрузки лишних, а с учетом необходимости использования графических материалов – еще и в значительной мере объемных сопроводительных документов, что в свою очередь повышает время экспорта и импорта данных, а также излишне занимает передающий канал, снижая, тем самым скорость эффективной передачи. Помимо вышеназванного, нельзя не отметить тот факт, что в рамках выпускной квалификационной работы сопроводительная расчетно-пояснительная записка должна хранится в формате с расширением doc, который является сложным текстовым форматом с большим количеством привязанных стилей и в связи с этим плохо поддается мониторингу версий с использованием стандартного инструментария гит. Для того, чтобы решить данную проблему, можно было бы использовать более простые текстовые форматы, такие как rtf или txt, однако данные текстовые форматы не соответствуют рекомендациям об оформлении сопроводительной расчетно пояснительной записки к выпускной квалификационной работе, а помимо этого еще и обладают довольно бедным перечнем опций для настройки текста, в связи с чем приведение внешнего вида работы в состояние, приемлемое для прохождение нормоконтроля становится затруднительным.

Принимая во внимание все вышесказанное, было принято решение поддерживание локального версионирования сопроводительных документов, при этом не чураясь выгрузки их в систему контроля версий. В результате данного подхода, содержимое папки с выпускной квалификационной работой и удаленного репозитория после синхронизации в полной мере аналогично, в чем можно убедиться изучив рисунки 7 и 8

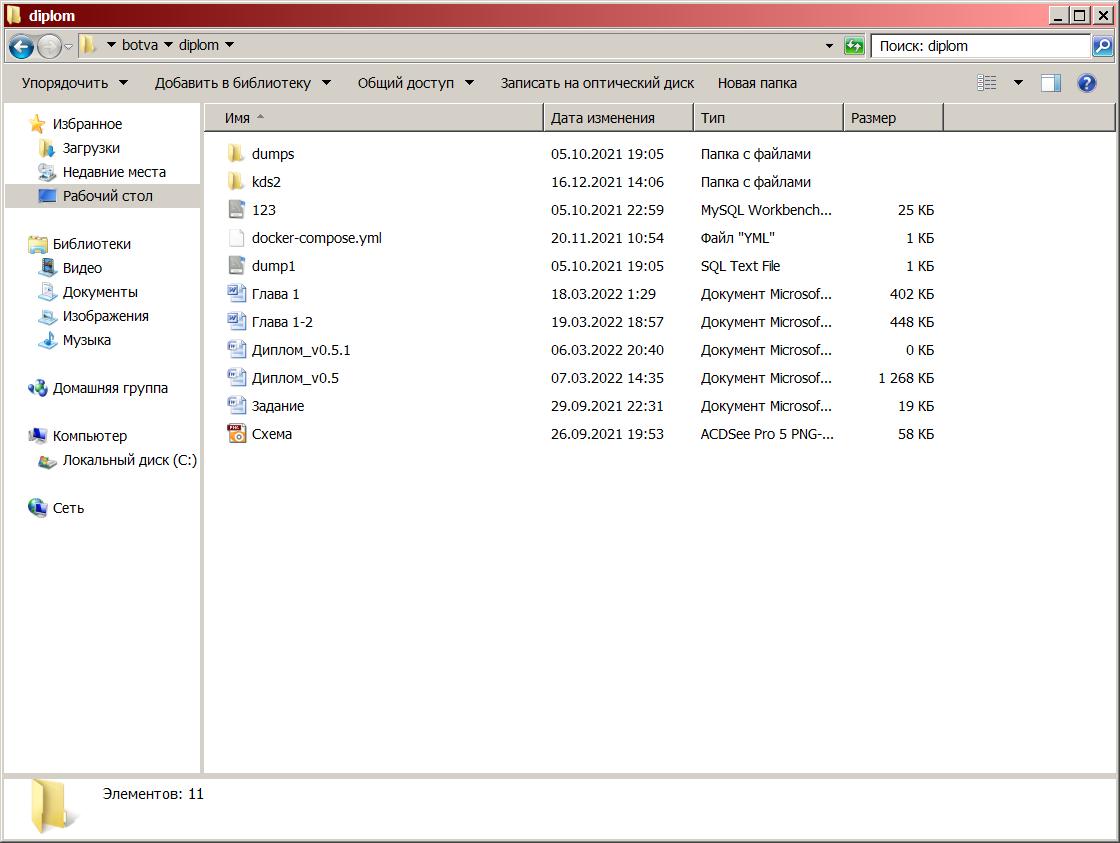
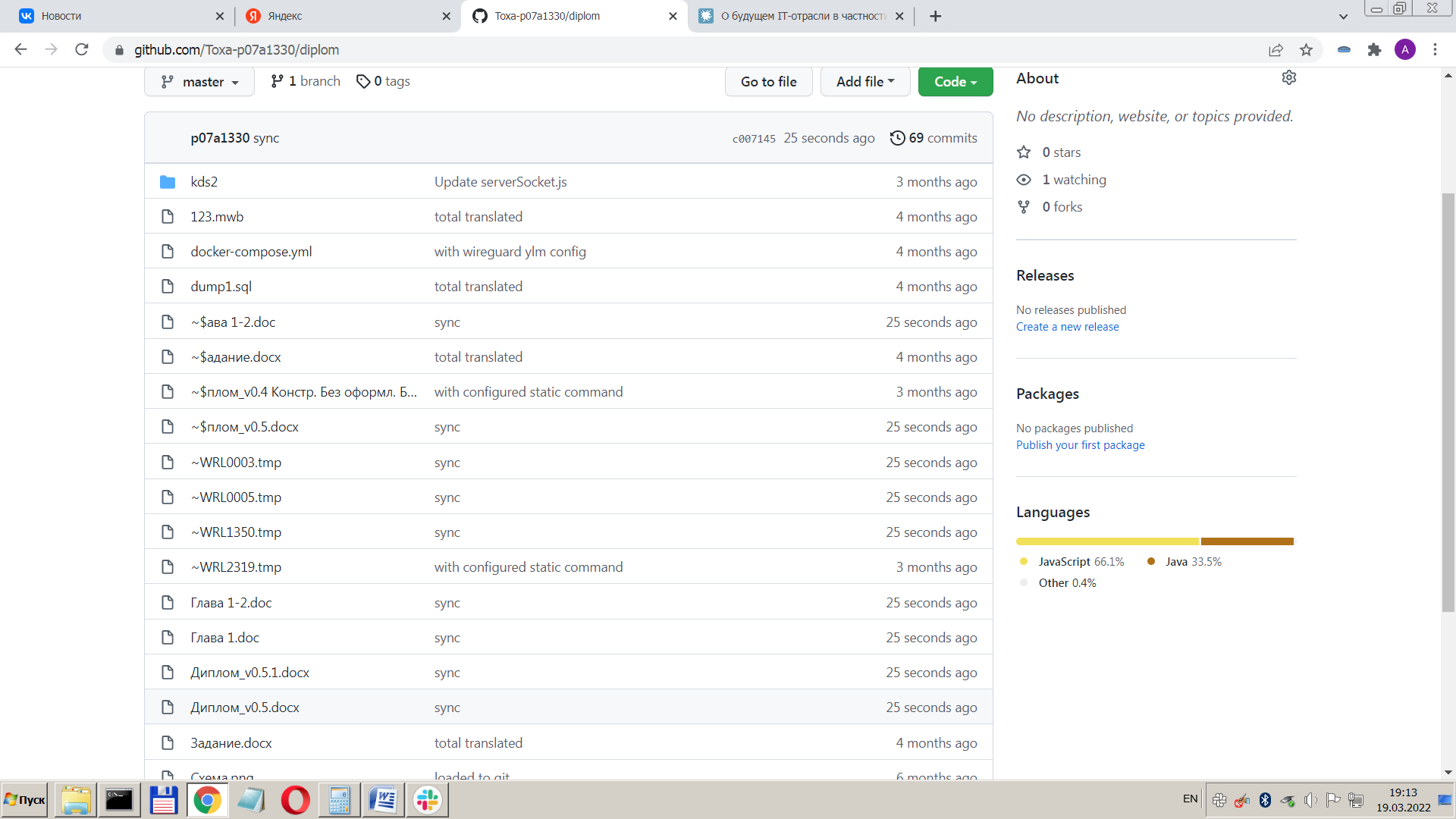


Рисунок 7 – отображение локальной папки с дипломом на момент написания

Рисунок 8 – отображение репозитория через глобальную систему контроля версий

Можно заметить, что на удаленном репозитории помимо файлов, видимых из локальной папки, есть также некоторое количество файлов с сломанной кодировкой в названии, а также файлов, которые имели место быть в данной папке раньше. Дело в том, что во первых, система контроля версий без тонкой настройки фильтров выгружает в репозиторий в том числе и временные файлы, а во-вторых, для того, что бы удалить те файлы, которые потеряли актуальность, нужно тоже предпринять ряд действий, отличных от стандартного поведения, предусматривающего выгрузку файлов с целью отслеживания изменений между версиями и создания резервной копии файлов. Безусловно, перед сдачей данного проекта или же перед передачей его на доработку третьим лицам, избыточные файлы должны быть подвергнуты удалению, однако во время активной разработки данные документы не оказывают какого-либо негативного влияния на рабочий процесс, в связи с чем могут быть оставлены без изменений.

### Выбор системы контроля версий

На сегодняшний день существует около десятка систем контроля версий, начиная с архаичных SCCS (система контроля версий исходного кода), заканчивая вполне актуальными git или mercurial. В данной выпускной квалификационной работе было принято решение использовать систему контроля версий Git, однако в целях обоснования данного решения должны быть рассмотрены также и альтернативные варианты. В частности, в данном разделе будут рассмотрены основные системы контроля версий, а также преимущества и недостатки каждой из них, а затем будет сделан вывод о том, какая из них является наиболее подходящей и почему в данном проекте надлежит использовать именно git.

RCS- «система управления пересмотрами версий.» - разработана в середине 80х. На сегодняшний день вытесянется системой контроля версий CVS, н тем не менее пользуется популярностью и активно используется, главным образом среди сообщества пользователей Linux. Данная система не позволяет работать с каталогами, создавая отдельную историю для каждого отдельного файла, поэтому она слабо подходит для проектов с большим количеством разобщенных исходных файлов. Кроме того, данная система не является многопользовательской – если какой-либо пользователь на данный момент времени вносит в файл изменения, в ней нету механизма, который позволяет редактировать данный файл кому0бы то ни было еще, а затем «слить» независимые изменения – файл просто блокируется для изменения другими пользователями системы.

Тем не менее, несмотря на вышеперечисленные недостатки. делающие ее непригодной для использования в данном проекте, данная система обладает и рядом преимуществ, которые позволяют ей оставаться используемой спусти почти полвека после разработки. В частности, к преимуществам данной системы принято относить простоту использования и реализации, низкие системные требования, высокую пригодность к резервном у копированию отдельных файлов, которые не требуют многожественного доступа (например, книга в процессе написания), а также широкая распространенность и предустановленность в существенной части свободного программного обеспечения.

Таким образом. исходя из вышесказанного, RCS является хорошей системой контроля версий, подходящей под решение большей части задач, которые могут быть перед ней поставлены, однако задача разработки системы управления банковскими терминалами бесконтактных платежей является слишком комплексной и объемной для столь легковесной системы, поэтому она не подходит.

CSV – система управления параллельными версиями представляет из себя развитее предыдущей системы.

CVS основана на клиент-серверой технологии, взаимодействующих по сети. При этом как клиент, так и сервер также могут располагаться на одном физическом компьютере, если над проектом осуществляет работу лишь один человек, или еть потребность вести локальный учет версий.

Работа CVS организована так: актуальная версия и все внесенные изменения хранятся в репозитории на сервере. Клиенты, при подключении к серверу, проверяют различие между локальной версией и серверной версий, сохраненной в удаленном репозитории, и, при наличии отличий, загружают их в локальный каталог. При необходимости осуществляется решение конфликтов и внесение требуемых изменений в продукт. После этого правки сохраняются в репозиторий удаленного сервера. CVS позволяет осуществлять переход на нужную версию проекта и осуществлять управление несколькими репозиториями одновременно.

К достоинствам данной системы принято относить возможность нескольким клиентам одновременно работать над единым проектом. Возможность управления проектом в целостности, а не отдельными файлами, наличие многочисленных удобных графических оболочек, широкая распространенность и отсутствие необходимости выгрузки файла целиком при создании текстовых изменений.

К недостаткам систем такого типа относят невозможность перемещения или переименования файла без утери истории изменений, сложности в поддержке нескольких веток одного проекта, невозможность частичной перезаписи бинарных файлов, низкую производительность и большое количество избыточных операций.

Резюмируя, можно сказать, что данная система в целом под поставленные задачи подходит, однако она морально устарела и не обеспечивает должной степени оптимизации по сравнению с современными конкурентами.

Subversion – эта централизованная система, была разработана в 2000 году и основана на клиент-серверной технологии. Ей присущи все достоинства CVS, при этом в ней решены основные проблемы поледней, о которых написано выше. Зачастую для обозначения этой ситемы используют аббервиатуру из названия ее клиентской части – SVN.

Способ взаимодействия с Subversion схож с работой через CVS. Клиенты осуществяют копирование изменений из удаленного репозитория и сливают их с локальным репозиторием пользователя. В случае возникновения конфликтов локальных записей и записей из репозитория, то требуется ручное их разрешение, после которого локальный проект изменяется должным образом, а результат выгружается в репозиторий.

При работе с документами, не позволяющими слияние изменений, можно использовать данный подход:

1. Файл загружается из репозитория и блокируется

2. Происходит внесение требуемых изменений.

3. Файл выгружается обратно в репозиторий и разблокируется

В значительной мере, про причине казуальности и сходства с CVS, Subversion успешно конкурирует с CVS и даже постепенно ее вытесняет.

Несмотря на это, и у Subversion есть ряд недостатков. Анализ преимушеств и недостатков данной сисетмы контроля версий представлен ниже.

Достоинства:

1. Схожая с CVS система команд и поддержка большинства ее возможностей.

2. Разнообразие и удобство как графических. Так и консольных интерфейсов.

4. Возможность отслеживания истории изменения файлов и каталогов в том числе после их перемещения или переименования.

5. Значительная производительность работы, как с двоичными, так и с текстовыми файлами.

6. Изначальная поддержка многими интегрированными средами разработки.

7. Возможность создания зеркальных копий репозитория.

8. Репозиторий поддерживает как работу с обычнми файлами, так и работу в режиме базы данных

9. Подробная и обширная документация

Недостатки:

1. Ввиду того, что н локлаьный компьютер выгружается полная версия репозитория, требуется существенное количество памяти на жестком диске.

2. Несмотря на то, что в отличии от CSV данная система поддерживает переименование файлов, при одновременном переименовании и загрузке одного и того же файла разными пользователями, могут возникнуть ошибки.

3. Не в достаточной степени проработан функционал ветвления и слияния веток.

4. Ввиду того, что информация, единоджы попавшая в репозиторий не может быть полностью удалена, могут возникнуть опеределенные проблемы с безопасностью.

В контексте выполнения выпускной квалификационной работы данная система уже имеет право на жизнь, так как она удовлетворяет всем возникающим потребностям и имеет весомые преимущества, в то время как ее недостатки в рамках данной работы не являются критичными. Точно также можно охарактеризовать такие системы как Aegis, Monotone или Mercurial – каждая из них относительно разнозначна и может быть применена, однако на сегодняшний день стандартом отрасли разработки программного обеспечения является выбранная мною система контроля версий Git. Анализ ее преимуществ и недостатков будет представлен ниже. Стоит также упомянуть, что на сегодняшний день насчитывается порядка десятка актуальных систем контроля версий и разбирать подробно преимущества и недостатки каждой из них – задача, безусловно, интересная, однако она несколько выходит за рамки данной работы и не представляет практической ценности.

Git – это гибкая, децентрализованная система контроля версий. Она в общем случае не имеет централизованного сервера и интересна в том числе и тем, что это первая система контроля версий, которая нашла свое применение не только в сфере разработки программного обеспечения. К примеру, Git используется в том числе для написания книг или – как в текущем случае – сопроводительных записок.

Каждый разработчик, использующий Git, имеет локальный репозиторий, который позволяет осуществлять локальный контроль версий. После этого, данные. Сохраненные в локальном репозитории изменения могут быть использованы для обмена с другими пользователями.

Несмотря на то, что Git является распределенной системой, зачастую при работе с ним создают центральный репозиторий, служащий для синхронизации. Версий между разработчиками. В качестве примера такого репозитория можно привести ядро операционной системы Linux, с которым можно ознакомиться по ссылке https://github.com/torvalds/linux

В таком случае каждый участник проекта осуществляет свои локальные доработки и без каких либо проблем скачивает обновления из корневого репозитория. Когда необходимые задачи отдельными разработчиками проекта решены и их решения прошли тесты и отладку. они, после проверки владельца центрального репозитория на предмет корректности и актуальности внесенных правок, вливают сделанные изменения в корневой репозиторий.

Наличие локальных репозиториев помимо всего прочего в значительной степени повышает надежность хранения информации, так как, в случае если один из них выйдет из строя, информация может быть в значительно степени восстановлена из других.

Работу над проектом в Git можно вести в множестве различных веток, которые после этого могут быть полностью или частично слиты, удалены, переключены на предыдущую версию или выкачаны как корень для независимого репозитория.

Достоинства:

1. В Git реализована надежная система для проверки данных и сравнения версий, на базе алгоритма хеширования SHA1.

2. Проработанная система слияния веток , напротив, расщепления версии на несколько раздельных веток (так называемый сборщик вишни).

3. Благодаря наличию локального репозитория, который содержит содержащего полную информацию о изменениях, можно осуществлять в полной мере локальный контроль ревизий и выгружать в корневой репозиторий только те версии, которые полностью прошли проверку изменения.

4. Хорошая степень оптимизации и высокая скорость работы

5.Проработанный и дружелюбный к пользователю перечень команд с подробной контекстной справкой.

6. Наличие удобного графического интерфейса, который позволяет решить большинство посталенных задач.

7. Интеграция в все современные интегрированные среды разработки.

8. Наличие подробной документации на многих языках.

Говоря о достоинствах, нельзя не упомянуть и недостатки:

1. Ориентированность на Unix системы. Поддержка Windows или MacOS есть, однако она появилась сравнительно поздно и до недавнего времени была доступна не полностью.

2. Есть пусть и крайне малая, но отличная от нуля вероятность совпадения хэш-кода у различных версий одного и того же проекта.

3. Нет функционала для отслеживания изменений отдельных файлов, что может привносить неудобства если такая потребность возникнет.

4. Крайне продолжительная первая загрузка масштябных репозиториев, ввиду необходимости скачивания всех правок, чье количество может бть весьма существенным. Например, ядро Linux имеет более 1000000 ревизий.

Таким образом, можно сказать, что Git – в максимальной степени мощная и удобная система контроля версий, которая способна решить все задачи, которые перед ней ставят, и именно поэтому она стала на сегодняшний день стандартом в данной сфере. По вышеперечисленным причинам, она была выбрана для управления версиями в данной работе.

Что касается сервиса для хранения корневго репозитория, выьор стоит намного менее широкий – на данный момент их по большому счету из сколь-либо масштабных, существует всего 2 - GitHub и GitLub. Однако ввиду того, что GitLab в значительной степени платный, выбора по большому счету не стояло – в рамках данной работы рациональным было использовать только GitHub.

## Технологическая часть

### 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. Пример записей в этом микросервисе приведен на рисунке 9.

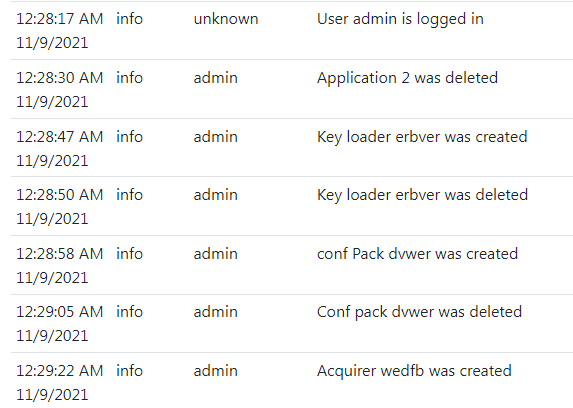


Рисунок 9 – пример отображения записей из таблицы логов

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 выпускной квалификационной 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 выпускной квалификационной 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 выпускной квалификационной работе не является окончательной, а представляет из себя скорее доказательство работоспособности концепта, который будет развиваться далее и достигнет своего апогея в рамках написания не бакалаврской, но магистрской работы. С одной стороны, нельзя не признать, что даже в рамках бакалаврской выпускной квалификационной 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 одного дпиломного 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 выпускной квалификационной 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. Работа VPN проилдюстрирована н рисунке 10.

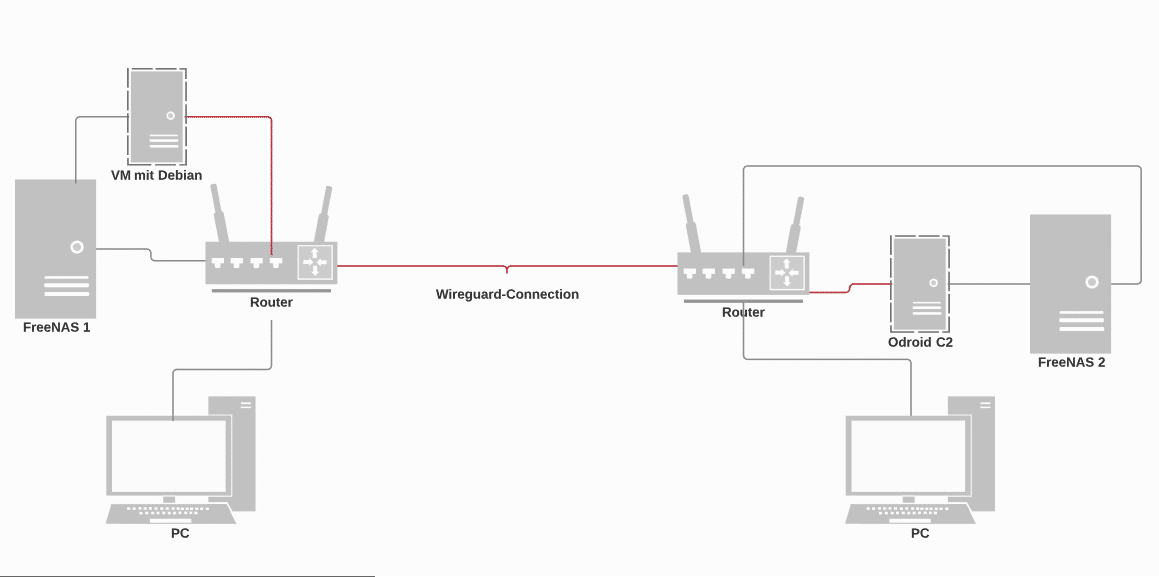


Рисунок 10 – принцип работы VPN туннеля

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, для которого назначается приватный ключ и IP адрес. После этого происходит загрузка конфигураций других пиров: их публичные ключи, IP адреса и т.д.

Каждый IP пакет, который приходит на WireGuard интерфейс инкапсулируют в UDP из-за чего его можно [безопасно доставлять](https://www.wireguard.com/protocol/) остальным пирам.

Клиенты определяют публичный IP адрес сервера в конфигурациях. Сервер автоматически получает внешние IP адреса клиентов, при получении от них корректных аутентифицированных данных.

Сервер может осуществить смену публичного IP адрес без прерывания работы. В результате этого он осуществит рассылку оповещения подключенным клиентам, которые в свою очередь и они осуществят обновление своей конфигурации в режиме реального времени.

При этом используется паттерн маршрутизации под названием [Cryptokey Routing](https://www.wireguard.com/#cryptokey-routing). WireGuard осуществляет прием и отправку пакетов на основании открытого ключа отправителя. После того как сервер расшифровывает корректный аутентифицированный пакет, осуществляется проверка его src поля. В случае, если оно соответствует с конфигурацией, происходит разрешение  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.

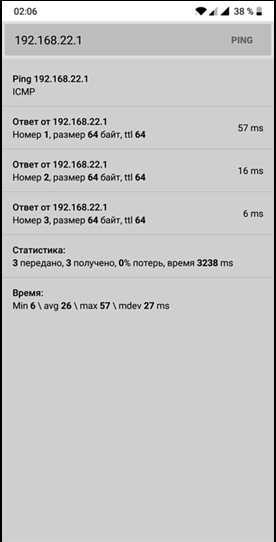
Для проверки работоспособности можно отправить ICMP пакет до сервера, обращаясь к нему по адресу в локальной сети, используя утилиту PingTool Network Utility. Как можно видеть из рисунка 11, пакет успешно доставлен.  


Рисунок 11 – 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. Различие между виртуализацией и контейнериацией представлено на рисунке 12.

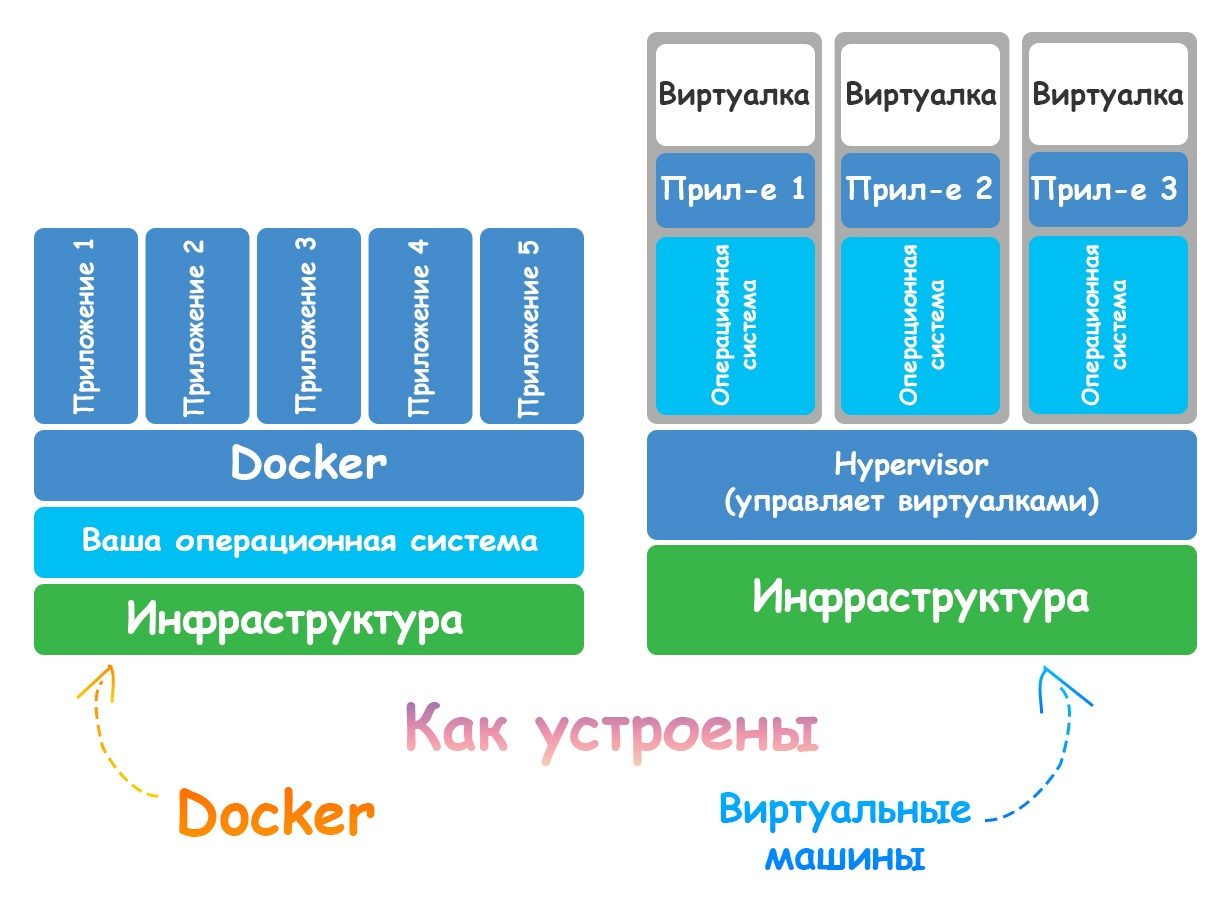


Рисунок 12– Различие между Докером и традиционными системами виртуализации

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. Сравнительная таблица наиболее популярных платформ приведена на рисунке 13.



Рисунок 13 – 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, на рисунке 14.



Рисунок 14 – главный экран тестировочного приложения

На рисунке 15 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.

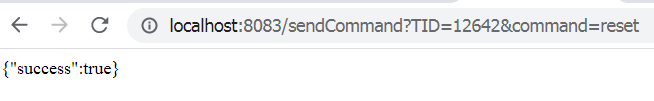


Рисунок 15 – Пример запроса из браузера   
При этом даже в текущей, базовой реализации такой запрос будет корректно обработан терминалом, что можно видеть исходя из рисунка ниже. Само собой, на данный момент в качестве обработки служит вывод пользовательских уведомлений, однако стоит понимать, что на момент выгрузки кода на терминал данный функционал будет расширен, в том числе методами, проприетарными для терминала. Пример отображения ответа сервера привеен на рисунке 16



Рисунок 16 – 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 выпускной квалификационной 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%, как можно видеть из рисунка 17.

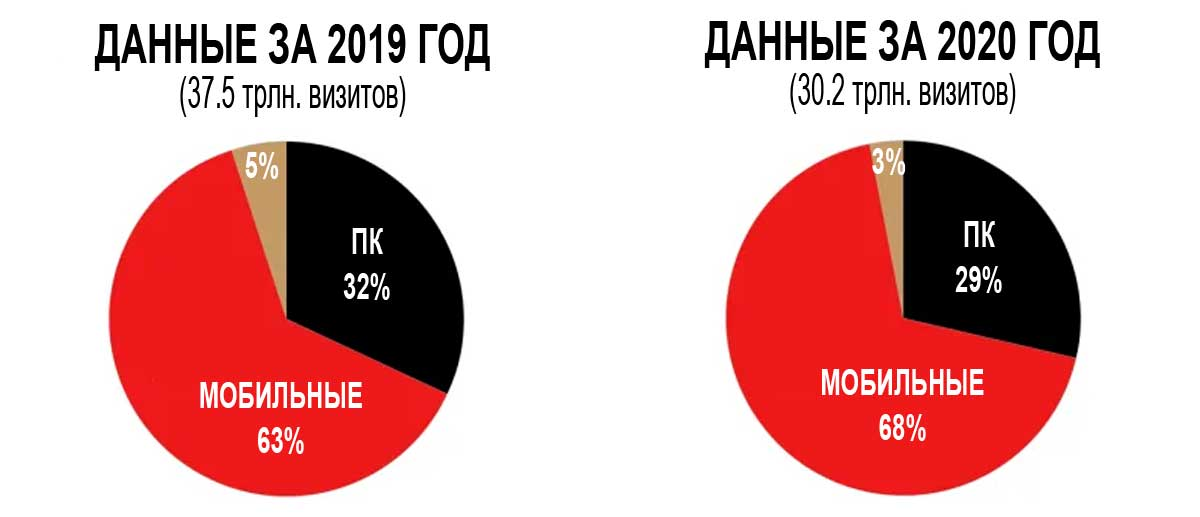


Рисунок 17 – соотношение между посещениями сайтов с персональных компьютеров и с мобильных телефонов

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

Осуществлена модернизация прокалочной камерной печи, заключающаяся в следующем:

— улучшена футеровка печи и дверцы загрузочного окна путем применения в качестве теплоизоляции материала материала волокнистого строения муллитокремнеземистого состава «Войлок МКРВХ-250» (Россия);

— для лучшей герметизации рабочего пространства печи дверца загрузочного окна сделана наклонной;

— замена нихромовых нагревательных элементов на силитовые стержни с целью создания стабильного температурного поля по сечению печи;

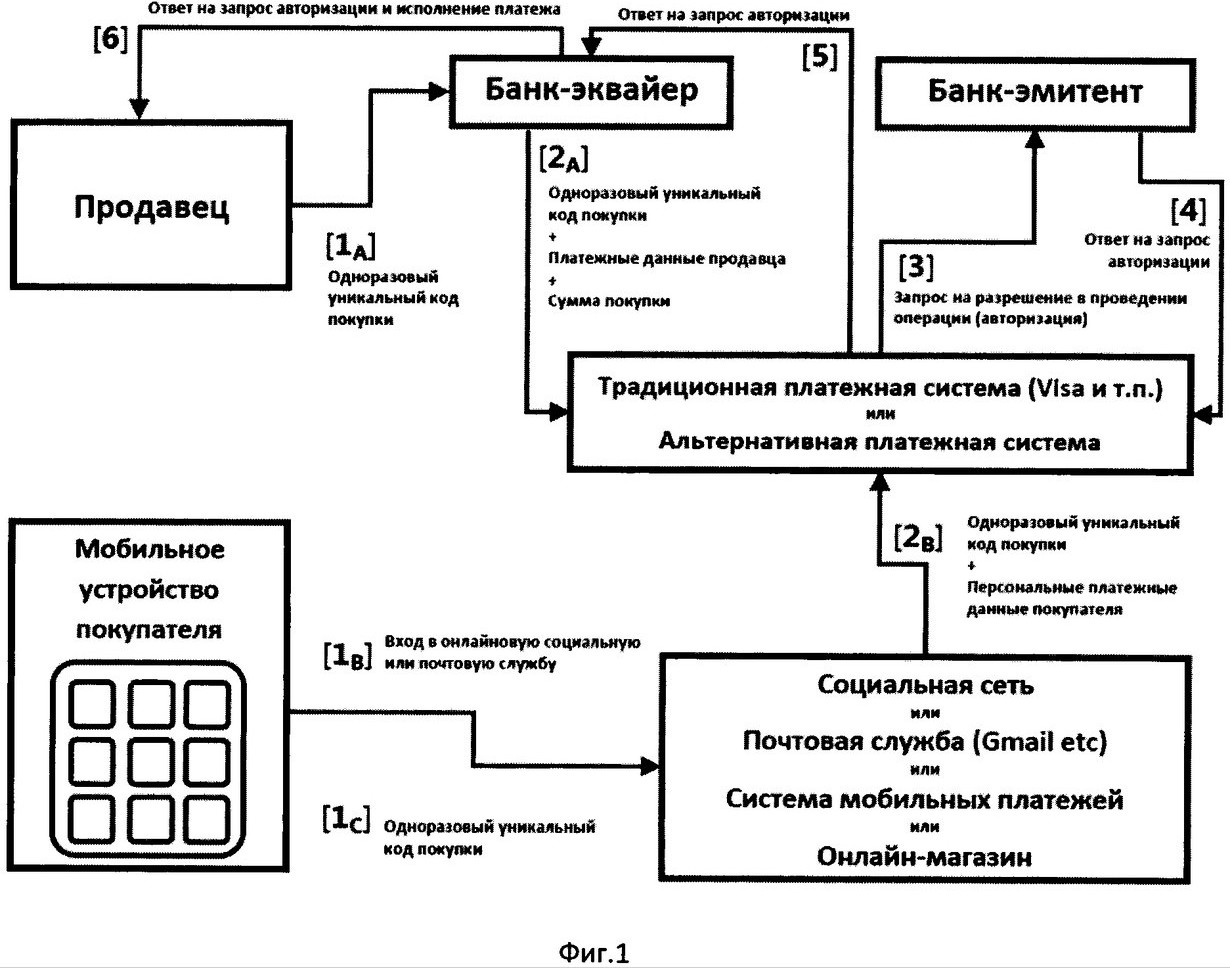
— подбор параметров силитовых нагревателей, расчет их количества и мощности печи (90 кВт, не менее);

— разработка схемы электропитания печи и элементов конструкции силитовых нагревателей и печи.

Приложение 1 – сравнительная таблица протоколов VPN

|  | **PPTP** | **SSTP** | **L2TP/IPsec** | **IKEv2/IPsec** | **OpenVPN** | **WireGuard** |
| --- | --- | --- | --- | --- | --- | --- |
| Компания-разработчик | Microsoft | Microsoft | L2TP — совместная разработка Cisco и Microsoft, IPsec — The Internet Engineering Task Force | IKEv2 — совместная разработка Cisco и Microsoft, IPsec — The Internet Engineering Task Force | OpenVPN Technologies | Jason A. Donenfeld |
| Лицензия | Proprietary | Proprietary | Proprietary | Proprietary, но существуют реализации протокола с открытым исходным кодом | GNU GPL | GNU GPL |
| Развертывание | Windows, macOS, iOS, некоторое время GNU/Linux. Работает “из коробки”, не требуя установки дополнительного ПО | Windows. Работает “из коробки”, не требуя установки дополнительного ПО | Windows,Mac OS X, Linux, iOS, Android. Многие ОС (включая Windows 2000/XP +, Mac OS 10.3+) имеют встроенную поддержку, нет необходимости ставить дополнительное ПО | Windows 7+, macOS 10.11+ и большинство мобильных ОС имеют встроенную поддержку | Windows, Mac OS, GNU/Linux, Apple iOS, Android и маршрутизаторы. Необходима установка специализированного ПО, поддерживающего работу с данным протоколом | Windows, Mac OS, GNU/Linux, Apple iOS, Android. Установить сам WireGuard, а затем настроить по руководству |
| Шифрование | Использует Microsoft Point-to-Point Encryption (MPPE), который реализует RSA RC4 с максимум 128-битными сеансовыми ключами | SSL (шифруются все части, кроме TCP- и SSL-заголовков) | 3DES или AES | Реализует большое количество криптографических алгоритмов, включая AES, Blowfish, Camellia | Использует библиотеку OpenSSL (реализует большинство популярных криптографических стандартов) | Обмен ключами по 1-RTT, Curve25519 для ECDH, RFC7539 для ChaCha20 и Poly1305 для аутентификационного шифрования, и BLAKE2s для хеширования |
| Порты | TCP-порт 1723 | TCP-порт 443 | UDP-порт 500 для первонач. обмена ключами и UDP-порт 1701 для начальной конфигурации L2TP, UDP-порт 5500 для обхода NAT | UDP-порт 500 для первоначального обмена ключами, а UDP-порт 4500 — для обхода NAT | Любой UDP- или TCP-порт | Любой UDP-порт |
| Недостатки безопасности | Обладает серьезными уязвимостями. MSCHAP-v2 уязвим для атаки по словарю, а алгоритм RC4 подвергается атаке Bit-flipping | Серьезных недостатков безопасности не было выявлено | 3DES уязвим для Meet-in-the-middle и Sweet32, но AES не имеет известных уязвимостей. Однако есть мнение, что стандарт IPsec скомпрометирован АНБ США | Не удалось найти информации об имеющихся недостатках безопасности, кроме инцидента с утечкой докладов АНБ касательно IPsec | Серьезных недостатков безопасности не было выявлено | Серьезных недостатков безопасности не было выявлено |

Приложение 2 – полная схема проведения платежа



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 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 системные данные, а также, 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 ссылок на файлы исходного кода

|  |  |
| --- | --- |
| 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 |