

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
«МОиПЭВМ»

Направление подготовки 09.03.04 – «Программная инженерия»
Профиль Программное обеспечение вычислительной техники и
автоматизированных систем

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ
РАБОТА МАГИСТРА**

на тему

**Синхронизация учётных данных пользователей интернет-ресурсов
и приложений**

Студент _____ Самсонкин Андрей Сергеевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролер _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Рецензент _____ Афонин А. Ю.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2023

Содержание

Введение	4
1 Анализ предметной области	8
1.1 Анализ моделей синхронизации учётных данных	8
1.2 Анализ менеджеров паролей	16
1.2.1 Обзор Dashlane	16
1.2.2 Обзор LastPass	17
1.2.3 Обзор KeePass.....	18
1.2.4 Обзор Google Password Manager.....	19
1.2.5 Обзор Apple Keychain	20
1.3 Анализ алгоритмов шифрования.....	21
1.3.1 Симметричные алгоритмы шифрования	21
1.3.2 Функция формирования ключей	23
1.4 Анализ функциональных требований	24
1.5 Анализ нефункциональных требования	25
1.6 Технологический стек	29
Выводы.....	31
2 Построение моделей и алгоритмов	32
2.2 Графическое представление алгоритма синхронизации учётных данных с беспарольной защитой.....	33
2.3 Графическое представление алгоритма регистрации устройства и передачи ключа	36
Выводы.....	37
3 Реализация моделей и алгоритмов	38
3.1 Программная реализация модели и алгоритмов синхронизации.....	38

	3
3.2 Тестирование реализации модели и алгоритмов синхронизации	57
3.3 Апробация и практическое применение модели и алгоритмов синхронизации.....	66
Выводы.....	69
Заключение	70
Список используемых источников.....	71
Приложение А. Акт о вреднении результатов магистерской диссертации	74
Приложение Б. Подробное описание прецедентов.....	76
Приложение В. Алгоритм регистрации устройства	80
Приложение Г. Алгоритм синхронизации учётных данных.....	88

Введение

Актуальность темы исследования. На данный момент большинство интернет-ресурсов и приложений требуют регистрацию для пользования и логин-пароль для аутентификации. Специалисты рекомендуют использовать разные пароли, чтобы при утечке данных на одном сервере под угрозой не были другие данные пользователя. При этом пароли должны быть достаточно сложными, чтобы их нельзя было просто подобрать.

Риск утечки в разы увеличивается, если учесть недавно обнаруженную уязвимость CVE-2021-44228 (log4j).

Многие компании, такие как Google, Facebook, Apple, VK, Яндекс вводят сервис аутентификации, где пользователь может подтвердить свою личность через сервер, не вводя пароль. Это очень эффективная тактика, так как снижает риск утечки учётных данных пользователя и облегчает доступ к интернет-ресурсам и приложениям. Но это подразумевает то, что разработчикам этих приложений необходимо вводить поддержку подобных сервисов. Это повышает затраты на разработку и поддержку, так как единого протокола пока не существует.

Системы управления учётными данными пользователей или менеджеры паролей позволяют создавать уникальные и сложные ключи для каждого веб-приложения. Это решает проблему с уникальностью, сложностью и запоминанием паролей. Но в настоящее время большинство пользователей получает доступ к интернет-ресурсам с нескольких разных устройств, например телефон, персональный компьютер, планшет. Поэтому рассматриваемое приложение должно работать на всех устройствах и синхронизировать между ними данные.

Данный механизм обычно реализуется через облако. Недостаток этого метода в том, что адрес этого сервера публичен, и он может быть атакован, из-за чего злоумышленники могут получить доступ ко всем учётным данным пользователя. Кроме того, владельцы облака потенциально сами могут сливать данные.

Чтобы решить эту проблему, данные должны храниться только на устройстве пользователя. Гипотетическая программа должна сделать этот процесс максимально простым для рядового пользователя.

Выполненные исследования опираются на результаты работ в области информационной безопасности таких экспертов и компаний как: Джереми Кейт, Avast, В.Б. Лим, А.М. Тереньтьев, А.А. Абидарова, М.А. Абдуллаева, Р. Мирзоев [1 - 6].

Объект исследования – система хранения и синхронизации учетных данных пользователей.

Предмет исследования – модель и алгоритмы хранения и синхронизации учётных данных пользователей.

Целью данной работы является уменьшение количества рисков кражи паролей пользователей и обеспечение удобства аутентификации в Интернете за счёт разработки модели и алгоритмов синхронизации и хранения учётных данных пользователей.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1) провести анализ методов обеспечения безопасности хранения и передачи учётных данных пользователей;
- 2) разработать графическую модель хранения и механизма синхронизации учётных данных пользователей интернет-ресурсов и приложений;
- 3) разработать алгоритмы синхронизации учётных данных пользователей интернет-ресурсов и приложений.

Решение поставленных задач позволит снизить риски кражи паролей и взлома учётных записей пользователей в Интернете.

Методы исследования. Основные результаты работы получены и обоснованы при помощи методов анализа и синтеза, системного анализа, теории защиты информации, технологий структурного и объектно-ориентированного программирования.

Достоверность научных результатов подтверждена теоретическими вкладами и результатами тестирования, а также сравнением полученных результатов с результатами, приведенными в научной литературе.

Научная новизна работы заключается в следующем:

- 1) разработана графическая модель хранения и передачи учётных данных, отличающаяся от существующих отсутствием в системе устройств третьих лиц, позволяющая хранить и передавать данные более децентрализованно;
- 2) разработан алгоритм синхронизации данных, отличающийся от существующих беспарольной системой защиты, и не позволяющий третьим лицам получить несанкционированный доступ к паролю пользователя;
- 3) разработан алгоритм регистрации устройств и передачи ключа, отличающийся от существующих наличием визуальной авторизацией и дополнительным подтверждением.

Основные положения, выносимые на защиту:

- 1) графическая модель хранения и передачи учётных данных без участия устройств третьих лиц;
- 2) графическое представление алгоритма синхронизации учётных данных с беспарольной защитой в нотации UML;
- 3) графическое представление алгоритма регистрации устройства и передачи ключа в нотации UML;
- 4) программное обеспечение, реализующее модели и алгоритмы синхронизации учётных данных пользователей.

Практическая и теоретическая значимость работы. Использование результатов исследования позволяет снизить риски утечки данных пользователя за счёт переноса данных с облака на устройство самого пользователя.

Апробация работы. Основные положения и результаты диссертационной работы докладывались и обсуждались на конференциях:

- 1) «Информационные технологии в науке и образовании. Проблемы и перспективы» в городе Пенза, ПГУ, 2021;

2) «Информационные технологии в науке и образовании. Проблемы и перспективы» в городе Пенза, ПГУ, 2022;

3) «Методы, средства и технологии получения и обработки измерительной информации» в городе Пенза, ПГУ, 2022;

4) «Информационные технологии в науке и образовании. Проблемы и перспективы» в городе Пенза, ПГУ, 2023.

Публикации. Основные результаты работы опубликованы в 4 печатных изданиях.

Структура и объем работы. Работа состоит из введения, 3 тематических глав, заключения, списка литературы и приложений. Общий объем основного текста – 73 страницы, включая 38 рисунка и 6 таблиц. Список литературы изложен на 3 страницах и содержит 23 наименований.

1 Анализ предметной области

1.1 Анализ моделей синхронизации учётных данных

Можно выделить 4 модели синхронизации учётных данных: клиент-серверную облачную, клиент-серверную пользовательскую локальную и с выходом в интернет и децентрализованную.

Клиент-серверная облачная модель является стандартной моделью, которая используется в большинстве имеющихся аналогов. Графическая модель представлена на рисунке 1.

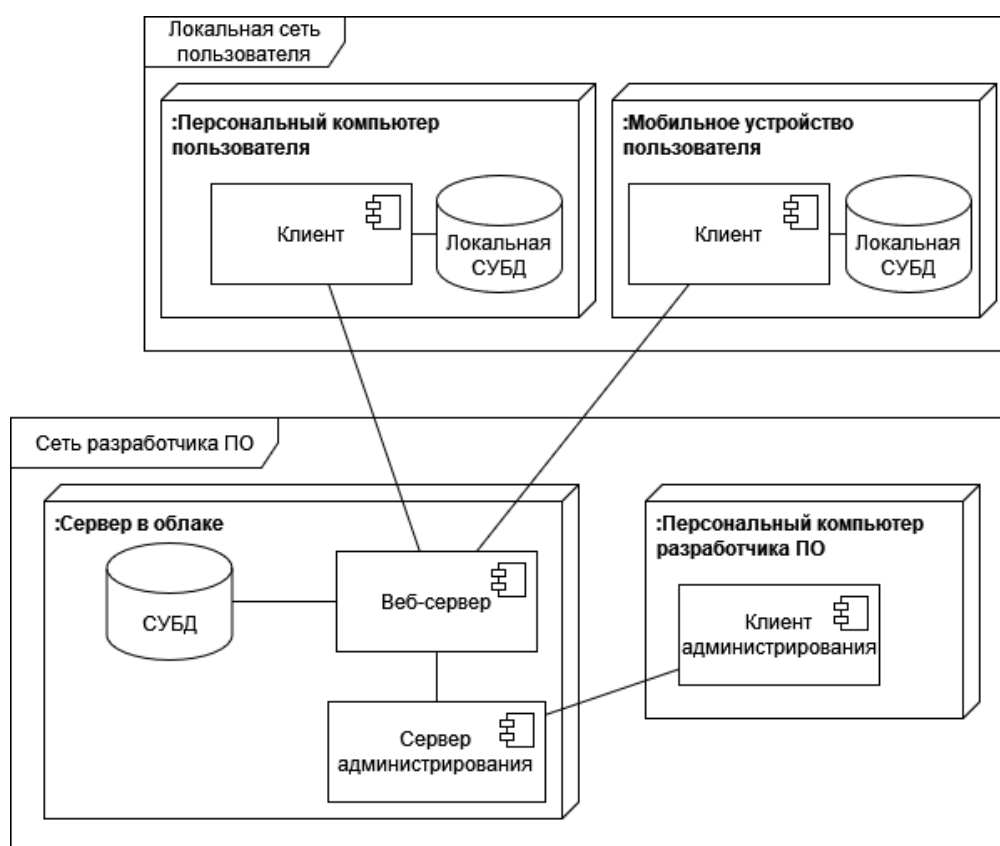


Рисунок 1 – Графическая модель клиент-серверной облачной системы

Данные хранятся на сервере в облаке и доступ к нему проходит через интернет. Сам сервер может находится где угодно, где есть выход в интернет. Адрес сервера публично известен. Для доступа к данным необходимо использовать логин-пароль. Так же доступна двухфакторная аутентификация. Передача данных защищена протоколом SSL. Может использоваться сквозное шифрование.

К преимуществам такой модели можно отнести простоту использования для пользователя (вся инфраструктура уже настроена компанией) и универсальность.

К недостатку можно отнести общую доступность сервера. Это и концентрация конфиденциальной информации разных пользователей делает её приоритетной целью для атаки. Всё это повышает риск утечки, особенно если сквозное шифрование не применяется.

Клиент-серверная пользовательская локальная модель отличается от пользовательской тем, что сервер находится в локальной сети пользователя. Таким образом все устройства, в которых хранятся конфиденциальная информация, принадлежат пользователю. Её схема представлена на рисунке 2.

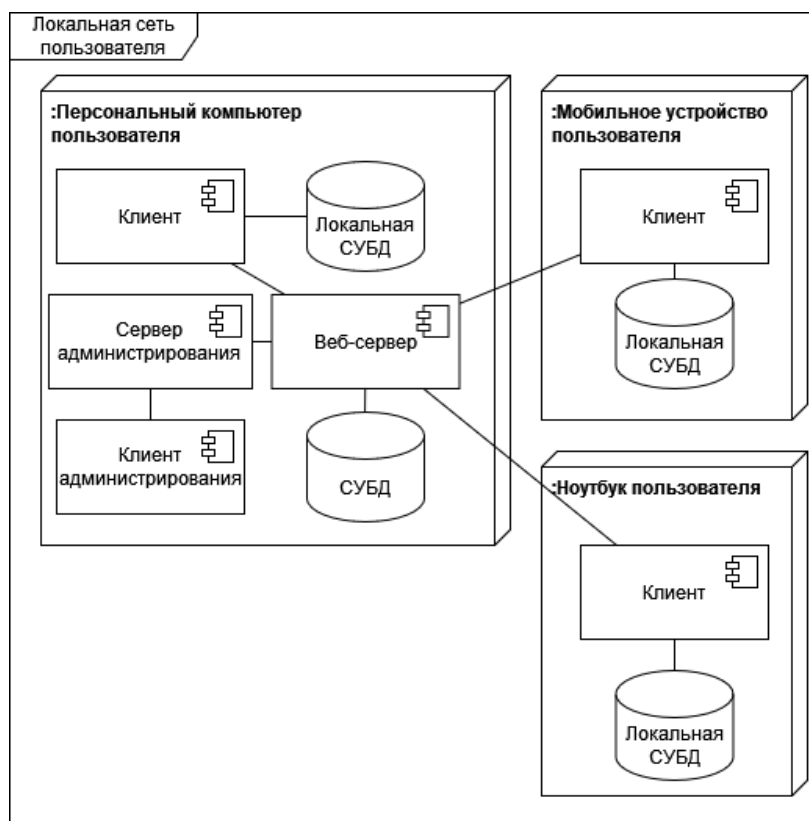


Рисунок 2 – Графическая модель клиент-серверной пользовательской локальной системы

Вся система может быть отрезана от интернета. Таким образом доступ к серверу невозможен из вне. Данные не уходят в сеть, что минимизирует риск утечки. Передача данных может быть полностью открытой, но для безопасности

рекомендуется использовать сквозное шифрование и шифровать весь трафик. Использовать SSL уже не получится, так как это потребует от пользователя собственного сертификата. Но этот протокол не нужен, так как у пользователя есть физический доступ ко всем устройствам и проблемы с обменом ключей возникнуть не должно. Также эта модель довольно проста в использовании. Всё что требуется, это установить приложение на потенциальный сервер, в качестве которого может выступать персональный компьютер.

Главным недостатком этой модели является её стационарность и изолированность. Чтобы поддерживать синхронизацию, каждое устройство должно быть в локальной сети, в которой находится конкретный сервер.

Модель с выходом в интернет является развитием предыдущей. Основным отличием является то, что сервер теперь доступен из интернета. Её схема представлена на рисунке 3.

Это хоть и избавляет её от главного недостатка, но возникают новые недостатки. Так, самый главный из них – теперь пользователю придётся самому настраивать всю инфраструктуру. Должен быть статичный IP-адрес, иначе каждый раз придётся менять настройки программы. Также пользователь сам должен открывать и пробрасывать порты. Резко повышается риск утечки данных, так как теперь сервер доступен извне, плюс персональные компьютеры обычно имеют меньшую степень защиты, чем промышленные сервера в облаках. С другой стороны IP-адрес пользователя не является публично известным. Возможная окупаемость атаки также под вопросом, так как личный сервер содержит конфиденциальную информацию в гораздо меньшем объёме, чем публичные сервера. Использование SSL тоже затруднительно, так как это требует регистрации сертификатов. Таким образом трафик должен шифроваться в самом приложении.

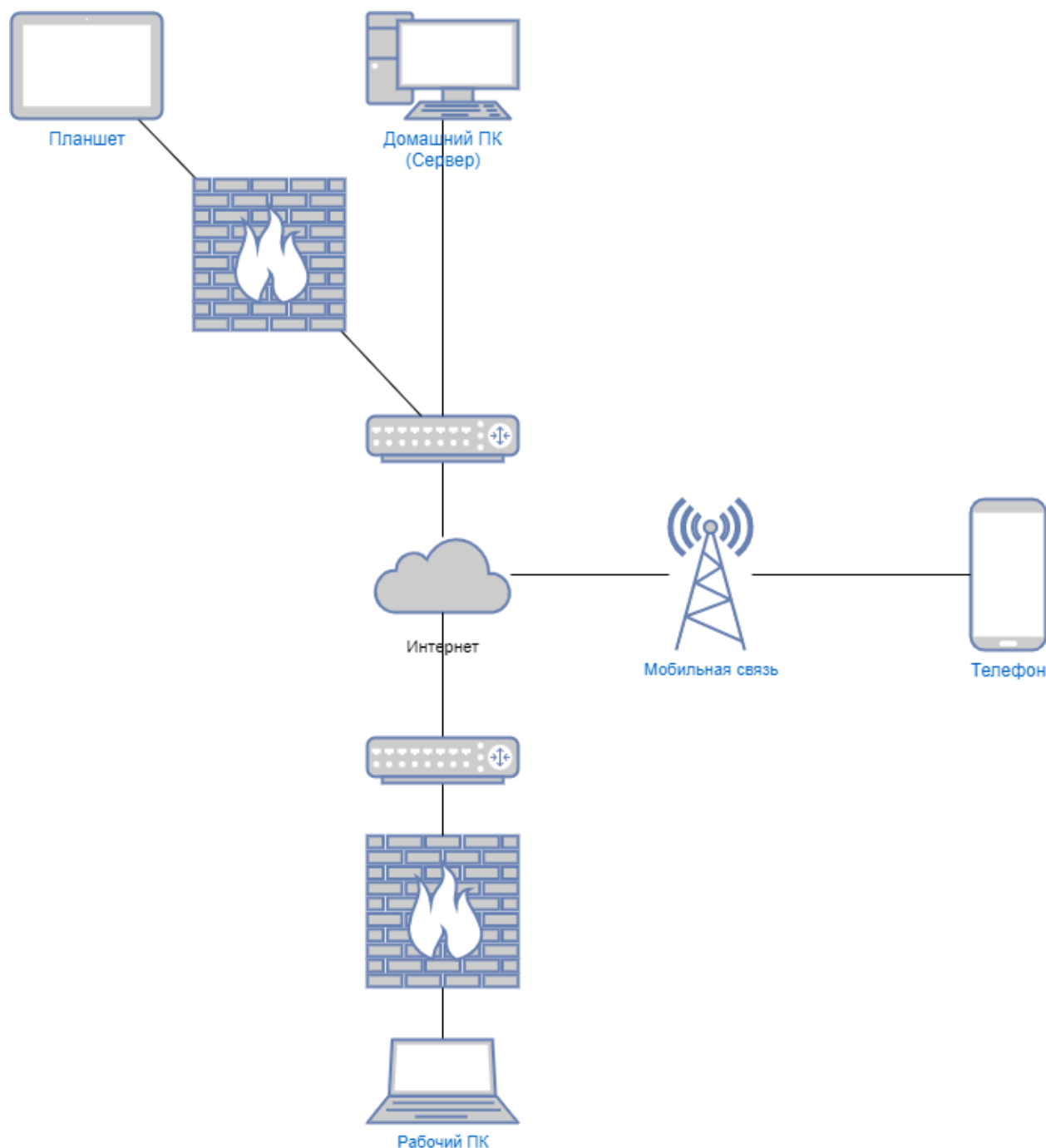


Рисунок 3 – Схема клиент-серверной системы с выходом в интернет

Децентрализованная модель является самой радикально отличающейся от всех остальных. Её схема представлена на рисунке 4. Главная её суть в том, что такое звено как сервер в принципе не существует. Все данные передаются между устройствами напрямую. Весь обмен данными происходит в локальной сети. Выход в интернет не используется. Таким образом снижается риск утечки данных. Для авторизации необходим физический доступ к одному из устройств. Протокол SSL не нужен, но обязательно иметь сквозное шифрование.

Синхронизация невозможна по интернету, но модель лишена главного недостатка клиент-серверной модели в локальной сети. В отличие от неё устройства не должны быть только в определённой локальной сети для передачи данных. Подойдёт любая сеть, где есть другое устройство с установленным приложением, чтобы осуществить передачу данных. Сами устройства могут выступать переносчиками данных на большом расстоянии. Модель крайне проста в использовании, так как не требует дополнительной настройки инфраструктуры.

Недостатками модели является вопрос в защите каналов связи и сложность её реализации (вплоть до того, что она нереализуема на мобильных устройствах Apple).

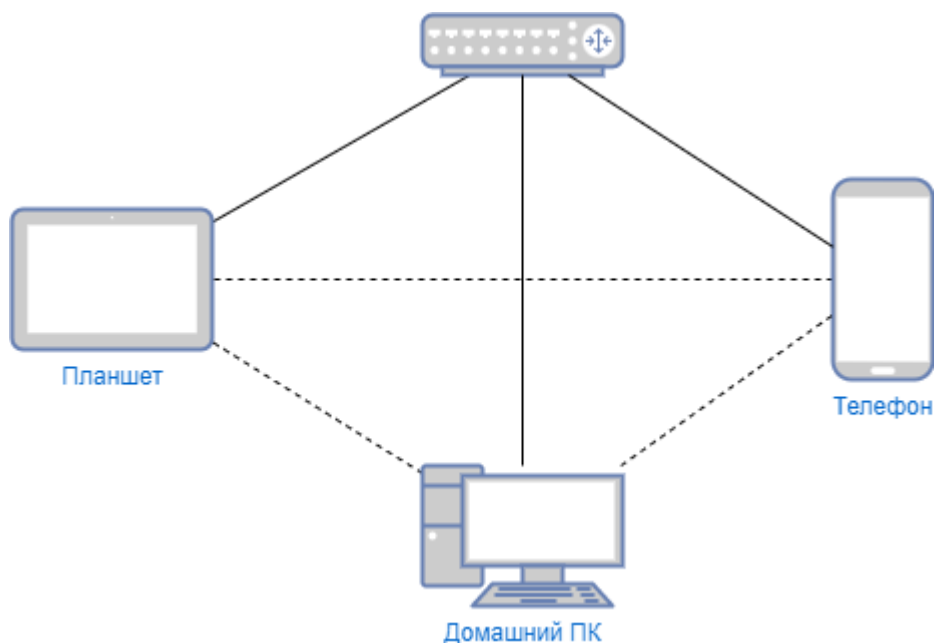


Рисунок 4 – Схема децентрализованной модели

Для того, чтобы сравнить приведённые модели, необходимо рассмотреть причины утечек паролей. Можно выделить следующие векторы атак:

- атака на сервер;
- атака на канал связи;
- атака на устройства пользователя;
- атака на конечные сайты и программы;
- атака на самого пользователя.

Атака на сервер, хранящий пароли, заключается в поиске уязвимостей в программно-аппаратном комплексе. Уязвимости могут быть программные и аппаратные. Обычно они связаны с несовершенством реализации.

Кроме атаки на уязвимости, сами работники сервиса могут сливать информацию ради получения прибыли

Защитой от подобных сливов может быть либо отказ от хранения данных на устройствах других людей, либо сквозное шифрование, которое позволяет читать информацию в незашифрованном виде только на устройстве, принадлежащем владельцу этих данных.

Кроме того, атаки могут проводиться по каналам связи. Например, атака посредника, которая позволяет злоумышленнику считывать трафик, отправляемый пользователем в интернет. Уже существует и применяется много средств для борьбы с подобной атакой, например HTTPS [7].

Также атаки могут совершаться на ресурсы, которыми пользуются сами пользователи. Особую опасность это представляет, если для разных ресурсов используются одинаковые пароли.

Личная информация может быть украдена и с устройств самих пользователей. В основном это происходит при заражении устройства программой-шпионом. Предотвратить это могут как системные средства защиты и антивирусы, так и сам пользователь своими действиями.

Обмануть можно и самого пользователя, например подсунув ему ложную форму, внешне ничем не отличающуюся от настоящей, которая отправляет данные злоумышленнику. Ситуация усугубляется, если украденный пароль используется множество раз.

Зная всё это, можно провести анализ приведённых моделей с точки зрения безопасности.

Для клиент-серверной облачной модели самым большим вопросом безопасности является сервер. Так как у него публичный адрес и он всем известен, безопасность данных зависит как от компетентности разработчиков, так и от компетентности злоумышленников.

Так как данные хранятся на оборудовании, которое не принадлежит пользователю, его владельцы могут получить доступ к этим данным и продать их.

Так же в программном продукте могут присутствовать недокументированные функции, особенно, если приложение имеет закрытый исходный код. Но даже с открытым исходным кодом нет гарантии, что приложение на сервере не является модификацией.

Подобные уязвимости являются платой за простоту использования и делегирование развёртки инфраструктуры и её поддержки третьим лицам.

В клиент-серверной локальной модели все устройства вплоть до сервера принадлежат пользователю. Поэтому продажа данных не является фактором. Более того, так как сеть закрыта и неизвестна, а данных на порядок меньше, атака на такую модель становится более трудоёмкой и менее рентабельной.

При этом синхронизации доступна лишь в пределах конкретной локальной сети. Но этот недостаток можно решить с помощью VPN, что требует от пользователя определённых навыков.

Клиент-серверная модель с выходом в интернет имеет те же достоинства относительно облачной, что и предыдущая. Однако синхронизация доступна во всей сети интернет. Это в свою очередь открывает новые уязвимости.

В таком случае развёртка инфраструктуры ложится на плечи рядового пользователя. Такая инфраструктура более уязвима к атакам из вне. Несмотря на это, она всё-ещё публично неизвестна и в ней всё-ещё малый объём данных, чтобы атака была рентабельной.

Кроме того, велика вероятность, что канал связи будет использовать протокол HTTP и на него можно произвести атаку посредника. Поэтому передаваемые данные должны быть зашифрованы.

У децентрализованной модели нет сервера. Поэтому многие атаки нереализуемы. Нет необходимости разворачивать инфраструктуру. Она работает на уже имеющейся.

Тем не менее у этой модели есть свои недостатки. Атака посредника всё ещё реализуема, если данные не зашифрованы. Кроме того, методы этой модели основаны на обнаружении устройств в сети, чем и может воспользоваться злоумышленник.

В таблице 1 приведено сравнение каждой модели и каждого подхода.

Таблица 1 – Сравнительные анализ моделей синхронизации и подходов хранения учётных данных.

Недостатки	Клиент-серверная облачная модель	Клиент-серверная локальная модель	Клиент-серверная модель с выходом в интернет	Децентрализованная модель	Запись на листочке
Публичный адрес	+	-	-	-	-
Много данных в одном месте	+	-	-	-	-
Продажа данных	+	-	-	-	-
Риск недокументированных особенностей (при открытом ПО)	+	-	-	-	-
Доступ извне	+	-	+	-	-
Только локальная синхронизация	-	+	-	-	-
MITM-атака	-	-	+	+	-
Обнаружение устройств	-	-	-	+	-
Уязвимость перед программами-шпионами	+	+	+	+	-
Сложность при использовании	-	-	+	-	-
Сложность при разработке	-	-	-	+	-
Уязвимость перед фальшивой формой	-	-	-	-	+
Отсутствие шифрования	-	-	-	-	+
Итого	6	2	4	4	3
Примерное время реализации (час)	144	112	144	336	0

Исходя из этого была выбрана клиент-серверная пользовательская модель, потому что она проста в реализации, использует исключительно устройства

пользователя и её легко масштабировать до модели с выходом в интернет, так как вторая является её расширением.

1.2 Анализ менеджеров паролей

Рассмотрим основные менеджеры паролей, такие как: Dashlane, LastPass, KeePass, Google Password Manager, Apple Keychain.

1.2.1 Обзор Dashlane

Dashlane – это условно-бесплатный кроссплатформенный проприетарный менеджер паролей и цифрового кошелька, разработанный одноимённой компанией. Dashlane поставляется как приложение, которое поддерживает Windows, macOS, iOS, Android.

Dashlane использует симметричный алгоритм AES256 для шифрования сохранённых паролей. Dashlane автоматически синхронизирует информацию на устройстве и отправляет зашифрованные данные в облако.

Dashlane использует мастер-пароль для шифрования сохранённых паролей на устройстве [8].

На рисунке 5 представлен интерфейс программы.

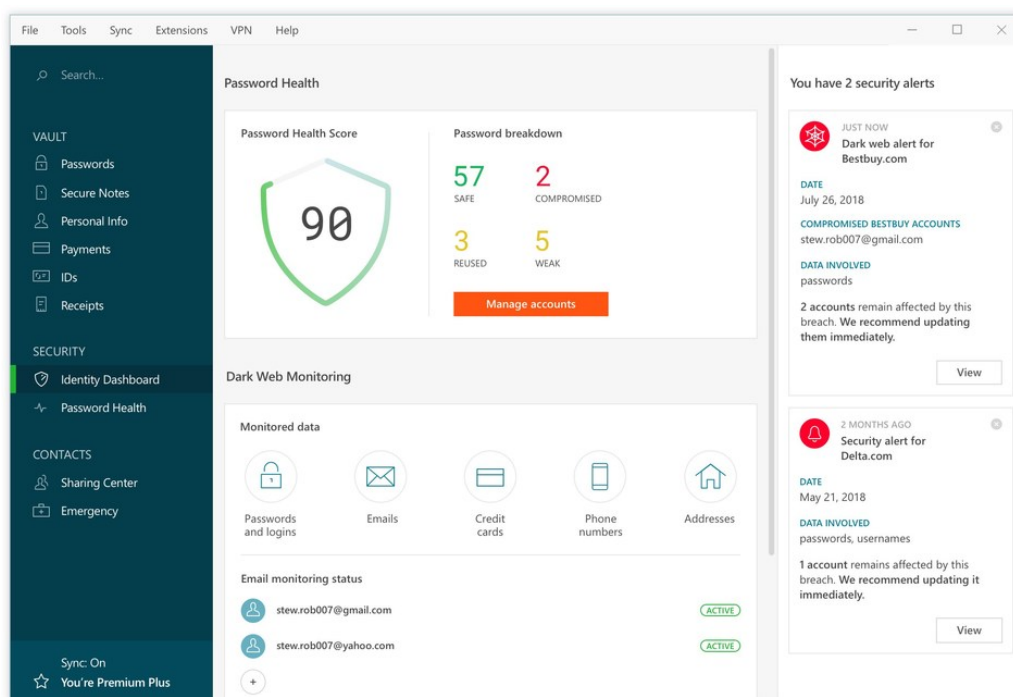


Рисунок 5 – Интерфейс Dashlane

Dashlane поддерживает такие функции, как:

- хранение паролей;
- генерация паролей;
- автоматическое заполнение форм;
- синхронизация с другими устройствами через облако;
- шифрование паролей на устройстве;
- цифровой кошелёк;
- многофакторная аутентификация (2FA, U2F);
- мониторинг утечек.

Dashlane предлагает широкий функционал по обеспечению безопасности данных. Он обеспечивает наилучшую безопасность из всех рассмотренных аналогов, которые поддерживают синхронизацию из облака.

Ключевым недостатком программы является её цена. Бесплатная версия идёт с существенными ограничениями в виде лимита на 50 паролей.

1.2.2 Обзор LastPass

LastPass – это условно-бесплатный кроссплатформенный проприетарный менеджер паролей, разработанный компанией LogMeIn. Он поставляется как плагин для браузеров Internet Explorer, Google Chrome, Mozilla Firefox, Opera, Safari или как приложение для macOS, iOS, Android [9].

LastPass использует алгоритмы AES256 для шифрования и PBKDF2 SHA-256 для хеширования [9].

Интерфейс обозреваемой программы представлен на рисунке 6.

Возможности LastPass:

- хранение паролей;
- генерация паролей;
- автоматическое заполнение форм;
- синхронизация с другими устройствами через облако;
- многофакторная аутентификация;
- общие пароли;

- общие папки;
- мониторинг утечек;
- зашифрованные заметки;
- зашифрованное хранилище данных.

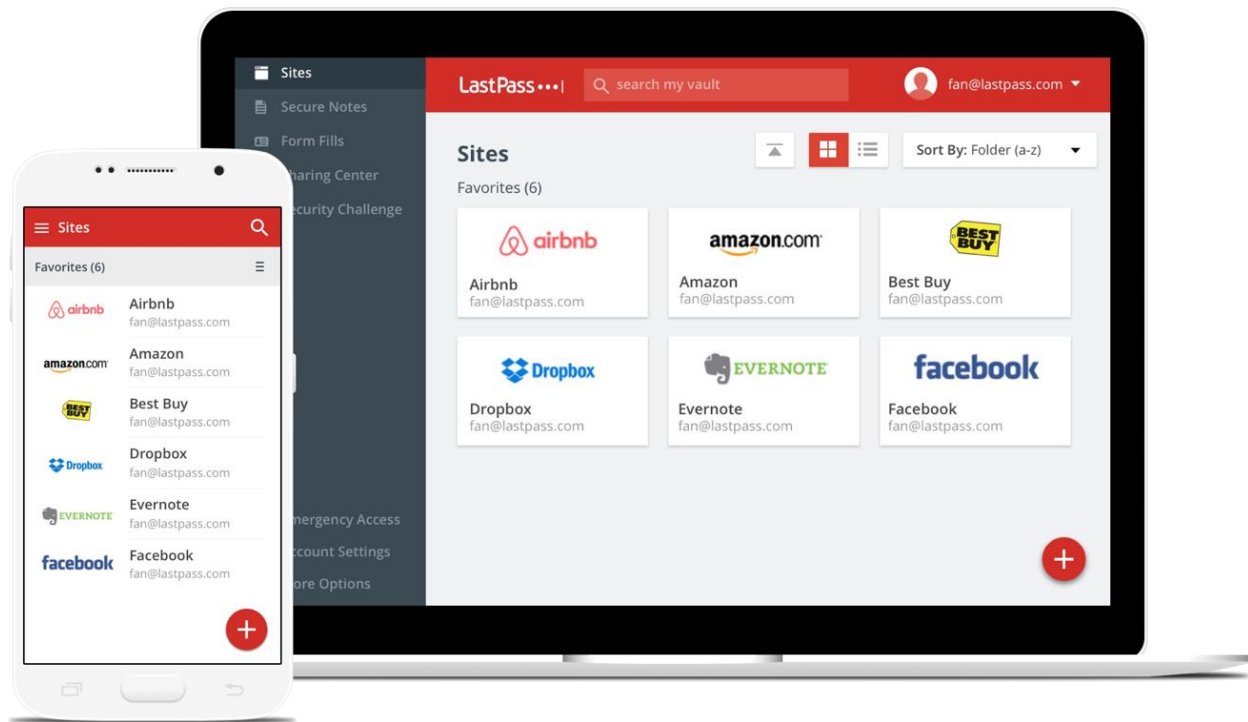


Рисунок 6 – Интерфейс LastPass

LastPass обладает широким функционалом, который доступен бесплатно. По сути он является более доступной альтернативой Dashlane с меньшими ограничениями в бесплатной версии. LastPass предоставляет ещё более широкий функционал при оплате годовой подписки.

За время жизни программы, было обнаружено большое количество уязвимостей и несколько утечек, что подрывает доверие к продукту [10].

1.2.3 Обзор KeePass

KeePass – это бесплатный кроссплатформенный менеджер паролей с открытым исходным кодом, разработанный Домиником Рейчелом. Он поставляется как приложение для основных настольных и мобильных платформ. На рисунке 7 представлен интерфейс программы.

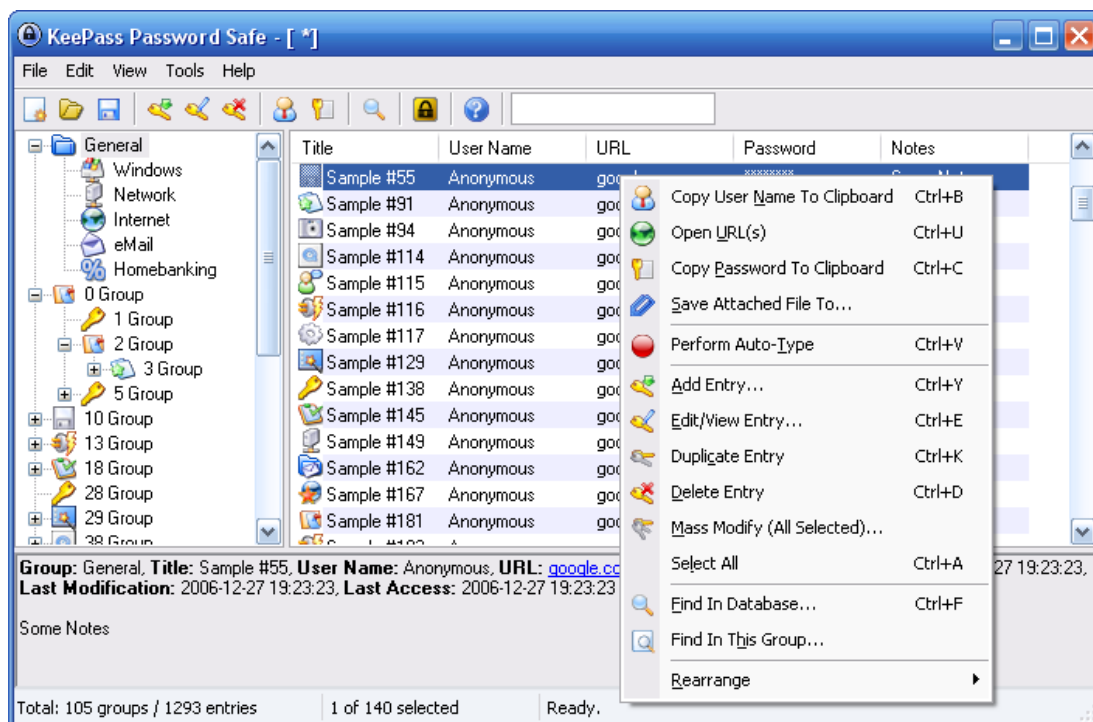


Рисунок 7 – Интерфейс KeePass

KeePass является портативным приложением для настольных платформ, написанным на C#. KeePass предлагает крайне широкий функционал абсолютно бесплатно. Он имеет открытый исходный код. Отказ от хранения информации облаком увеличивает сохранность паролей.

Недостатком KeePass является то, что этот продукт рассчитан на более опытных пользователей, по сравнению с аналогами. Множество функций, которые упрощают пользование программой не реализованы [11].

1.2.4 Обзор Google Password Manager

Google Password Manager – это бесплатный проприетарный менеджер паролей. Он работает только в браузере Chrome.

Google Password Manager является самым базовым и самым доступным решением из всех. Ключевым недостатком является то, что он работает исключительно в браузере и на ОС от Google.

1.2.5 Обзор Apple Keychain

Apple Keychain – это бесплатный менеджер паролей с открытым исходным кодом. Он работает только на устройствах от Apple. Это решение по умолчанию включает двухфакторную аутентификацию, как часть входа в аккаунт iCloud. На рисунке 8 представлен интерфейс программы.

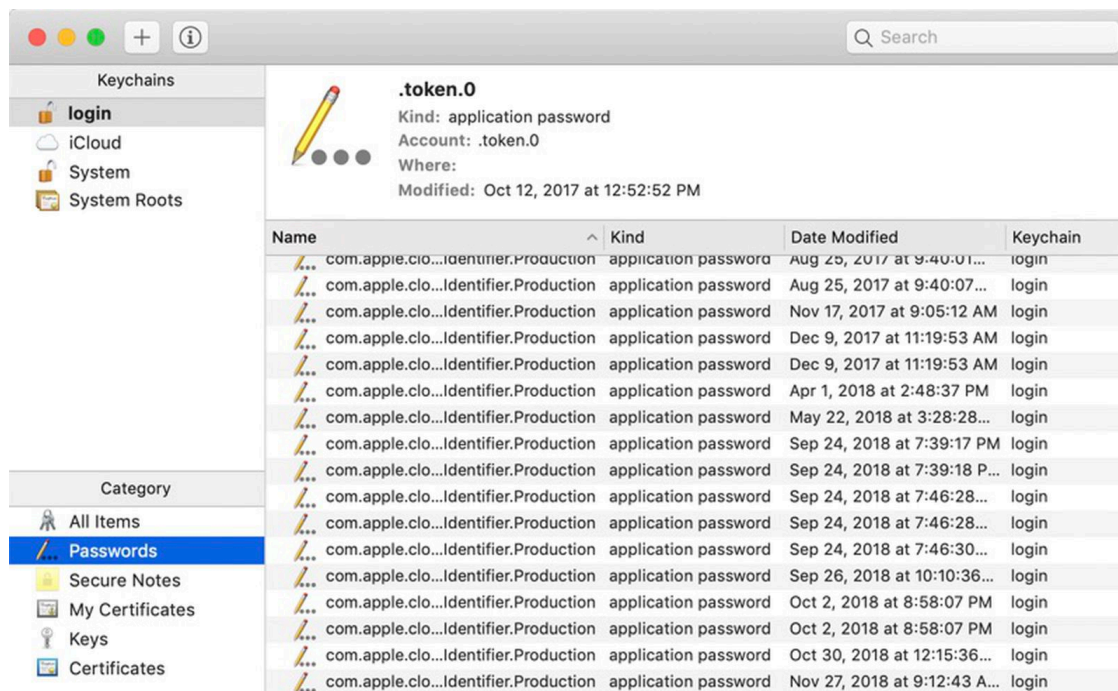


Рисунок 8 – Интерфейс Apple Keychain

Apple Keychain использует Triple DES для шифрования данных на устройстве.

Преимущества этого решения. Apple имеет хорошую репутацию в сфере информационной безопасности. Keychain отлично работает на всех устройствах Apple. Triple DES является менее популярным алгоритмом шифрования [12].

Ключевым недостатком Apple Keychain является то, что программа работает исключительно в рамках своей экосистемы.

В результате исследований были рассмотрены основные программы для хранения паролей и произведен их сравнительный анализ. Результаты приведены в таблице 2.

Таблица 2 – Сравнительный анализ программ для администрирования учётных записей пользователей

Название продукта	Поддерживаемые платформы	Модель распространения	Ограничения
Dashlane	Firefox, Chromium*, Safari Android, iOS	Условно бесплатная (от 300руб./мес.)	В бесплатной версии до 50 паролей, не более 1 устройства
LastPass	Firefox, Chromium*, Safari Android, iOS	Условно бесплатная (от 269руб./мес.)	Нет
KeePass	Windows, Linux, macOS, Android, iOS	Бесплатная	Нет автозаполнения для настольных браузеров Нет синхронизации по умолчанию
Google PM	Chrome, Android	Бесплатная	Только в экосистеме Google
Apple Keychain	Safari, macOS, iOS	Бесплатная	Только в экосистеме Apple
KeeWord	Windows, Android, iOS	Бесплатная	Синхронизация только в локальной сети

*Все браузеры на базе Chromium (Chrome, Opera, Edge, Yandex Browser и др.)

Каждое из рассматриваемых решений обеспечивает достаточный уровень защиты хранения учётных записей пользователей.

1.3 Анализ алгоритмов шифрования

1.3.1 Симметричные алгоритмы шифрования

Симметричные алгоритмы шифрования – это алгоритмы, в которых для шифрования и дешифрования применяется один и тот же криптографический ключ. Ключ должен сохраняться в тайне. Спецификации алгоритмов представлены в таблице 3.

Таблица 3 – сравнительная характеристика симметричных алгоритмов шифрования

	TripleDES	AES	«Кузнечик»
Длина ключа, бит	168	128/192/256	256
Длина блока, бит	64	128	128
Число раундов	48	10/12/14	9

TripleDES.

TripleDES (Triple Data Encryption Standard) – симметричный блочный шифр, созданный Уитфилдом Диффи, Мартином Хеллманом и Уолтом Тачманном в 1978 году на основе алгоритма DES с целью устранения главного недостатка последнего – малой длины ключа (56 бит). Размер ключа в 3 раза больше (168 бит). Из-за этого алгоритм медленнее своего предшественника в 3 раза.

TripleDES является менее популярным алгоритмом, чем AES, что делает его менее важной целью для атаки. Так же известных криптографических атак на 3DES не существует. Алгоритм реализован на множестве языков, что сводит к минимуму возможность атаки на ошибки реализации.

TripleDES постепенно выходит из употребления. Он обладает меньшей криптостойкостью и меньшей скоростью по сравнению с AES [13]. Так же размер ключа в 168 бит делает его формирование довольно проблематичным.

AES.

AES (Advanced Encryption Standard; Rijndael) – это симметричный алгоритм шифрования. Размер блока – 128 бит, ключ – 128/192/256 бит. Принят в качестве стандарта шифрования правительством США.

К достоинствам AES можно отнести криптостойкость, быстродействие, популярность. Благодаря последнему пункту, реализация алгоритма присутствует в стандартных библиотеках многих языков программирования. Кроме того, во многих процессорах существует аппаратная поддержка этого алгоритма. Благодаря этому, алгоритм менее подвержен атакам, направленным на ошибки в реализации алгоритма [14].

Недостатки исходят из популярности. Так как больше всего внимания уделено этому алгоритму, то больше всего сил направлено на поиск атак на него. Доказано, что некоторые виды алгоритма не устойчивы к атакам по сторонним каналам и уже существуют реализации подобных атак как известные, так и те, которые ещё не раскрыты [15, 16].

«Кузнечик» (ГОСТ 34.12-2018; RFC 7801) – симметричный алгоритм блочного шифрования с размером блока 128 бит и длиной ключа 256 бит, использующий для генерации раундовых ключей SP-сеть.

Сейчас очень мало информации об этом алгоритме, что одновременно является его достоинством и недостатком. Практически применимых атак для этого алгоритма пока не существует. Но он уязвим к атаке «встречи посередине» на 5 раундов и атакам по сторонним каналам [17].

Другим недостатком является непопулярность этого шифра. Из-за этого у него мало реализаций на разных языках программирования. Поэтому высока возможность атаки на ошибки реализации.

У «кузнечика» нет аппаратного ускорения в отличие от AES, поэтому он будет отставать в скорости от последнего.

1.3.2 Функция формирования ключей

Функция формирования ключей – это функция, формирующая один или несколько секретных ключей на основе секретного значения с помощью псевдослучайной функции. Обычно такие функции являются необратимыми. Полученные секретные ключи используются для шифрования данных.

«bcrypt» – криптографическая хеш-функция формирования ключа, для защищенного хранения паролей, основанная на шифре Blowfish. Для защиты от атак с помощью радужных таблиц использует соль (salt). Время работы функции можно настроить, чтобы усложнить атаку перебором, таким образом она является адаптивной.

«bcrypt» является довольно популярной и реализована во многих языках программирования, таких как Java, Python, C#, Ruby, Perl, PHP 5.3, Node.js, Go...

Алгоритм критикуют за уязвимость к перебору с помощью устройств ASIC и тот факт, что алгоритм расширения ключа мало исследован [18].

PBKDF2 (Password-Based Key Derivation Function; RFC 2898) – стандарт формирования ключа на основе пароля. Рекомендован в качестве замены

«bcrypt». Функция позволяет использовать произвольную псевдослучайную функцию (чаще всего используется HMAC).

Выявленным недостатком алгоритма считается его низкая пространственная сложность по сравнению с bcrypt и scrypt, что позволяет подбирать ключи на специальном оборудовании или графических ускорителях [19].

«scrypt» – адаптивная криптографическая функция формирования ключа на основе пароля. В отличие от «bcrypt» и PBKDF2, у scrypt можно регулировать не только количество итераций, но и пространственную сложность. Таким образом усложняется атака перебором при помощи ПЛИС и ASIC.

«scrypt» не так хорошо распространена, как её аналоги, поэтому у неё меньше реализаций на разных языках программирования. Высокая пространственная сложность может выступить серьёзным недостатком на устройствах конечного пользователя [19].

Таким образом можно сказать, что в качестве алгоритма симметричного шифрования предпочтительнее использовать AES. TripleDES уступает в скорости и надёжности своим более новым аналогам, а у «Кузнечика» нет аппаратной реализации на потребительской электронике.

С функциями хеширования паролей всё сложнее и нельзя однозначно сказать, какой алгоритм лучше. У каждого есть свои преимущества и недостатки в разных ситуациях. Можно комбинировать способы. Был выбран PBKDF2, так как он достаточно новый и его реализация уже есть в стандартных библиотеках.

1.4 Анализ функциональных требований

Программа должна выполнять следующий функционал:

- сохранять учётные данные;
- обновлять учётные данные;
- удалять учётные данные;
- очищать учётные данные;
- регистрировать устройство с установленным приложением;

- серверное приложение должно запускаться на ОС Windows 10 64-bit;
- приложение должно использовать локальное хранилище для доступа к данным;
- время реакции системы на действия пользователя должно составлять не более 100 миллисекунд;
- программа должна занимать не более 256 Мбайт на жестком диске;
- передаваемые при синхронизации данные должны быть зашифрованы;
- запущенная программа должна использовать не более 32 Мбайт оперативной памяти.

На данном этапе были разработаны функциональные и нефункциональные требования к системе [20].

Пользовательские интерфейс должен быть интуитивно понятен, чтобы пользователь мог эффективно решать свои задачи сразу после запуска программы без необходимости изучать её или читать инструкции.

Пользовательский интерфейс должен выполнять следующие требования:

- быть узнаваемым;
- быть привлекательным;
- быть интуитивно понятным;
- способным удержать внимание пользователя.

Под узнаваемость интерфейс должен быть схож с интерфейсом программ-аналогов. В этом случае пользователю будет легче начать работать и привыкнуть, если он уже имел дело с программами подобного рода.

Когда пользователь просматривает приложения в библиотеке приложений, он обращает в первую очередь внимание на оценку, иконку и скриншоты программы. Пользователь скорее всего остановится на программе, которая выглядит по скриншотам более современно чем аналоги.

Пользователь должен понимать, как работает программа лишь взглянув на интерфейс. Таким образом интерфейс должен одновременно выполнять роль инструкции. Чтобы достичь этого, необходимо использовать узнаваемые иконки

и пиктограммы. Это поможет пользователю быстрее ориентироваться в интерфейсе. Иконки должны быть однозначными и сильно отличаться друг от друга. У пользователя не должно возникать вопроса «А что делает эта кнопка?». Также у хорошего интерфейса должно присутствовать так называемое «пустое состояние». Это момент самого начала работы, когда пользователь еще ничего не сделал и интерфейс в это время должен подсказывать, с чего ему лучше начать.

Интерфейс должен акцентировать внимание на самом важном. Это можно сделать многими способами. Привлечение внимания цветом, пожалуй, один из самых эффективных. На экране должно присутствовать ограниченное количество элементов. Иначе большое нагромождение вызовет усталость. Заполнение форм можно разделить на несколько страниц.

Исходя из вышеперечисленного, интерфейс программы должен включать в себя страницы для следующих сценариев:

- пользователь запустил программу;
- пользователь просматривает хранилище;
- пользователь создаёт/редактирует учётные данные;
- пользователь открыл окно настроек.

На рисунке 10 показан список учётных записей. Это окно открывается после авторизации или при клику на раздел «Storage» в самой нижней панели. По нажатию на «+» можно добавить новую учётную запись. Кнопка «Edit» позволяет выбрать несколько учётных записей и проводить над ними совместные операции. Поле «Search» позволяет найти конкретную учётную запись по идентификатору или имени пользователя. Долгое нажатие по элементу списка или тап по трём точкам вызывает контекстное меню, где пользователь может скопировать имя пользователя или пароль, а также удалить учётную запись. Клик по элементу списка открывает меню настроек учётной записи, которое показано на рисунке 11.

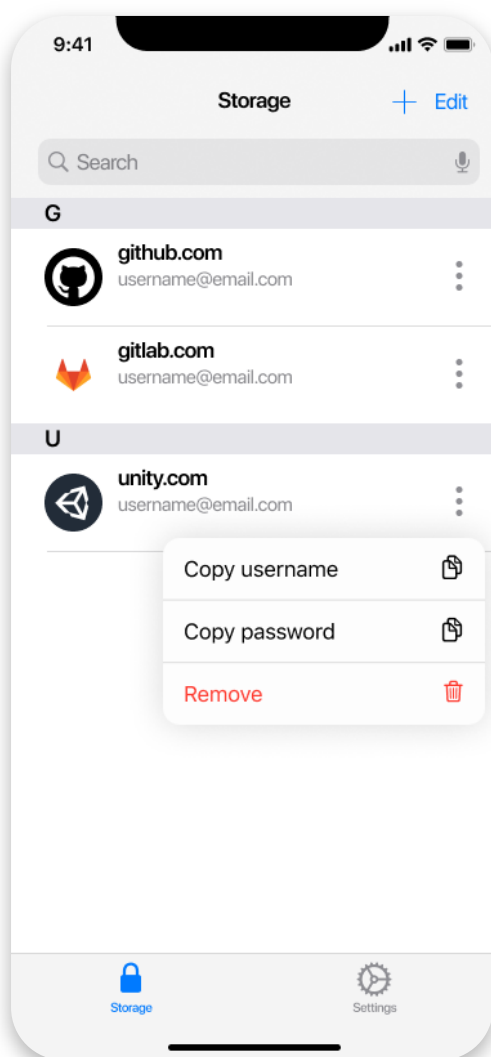


Рисунок 10 – Список учётных записей

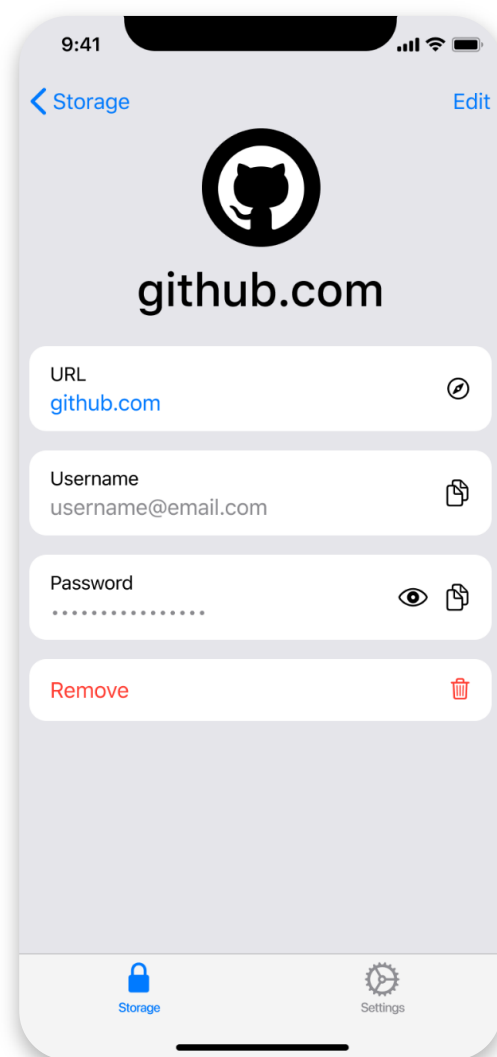


Рисунок 11 – Настройки учётной записи

В меню настроек учётной записи представлены главные поля каждого элемента. По клику на URL можно перейти на сайт. Имя пользователя и пароль можно скопировать, нажав соответствующую кнопку. Также пароль можно отобразить, нажав на иконку глаза. Чтобы удалить учётную запись, достаточно кликнуть на кнопку «Remove». Чтобы непосредственно изменить поля, необходимо включить режим редактирования через кнопку «Edit» в верхнем углу. По нажатию на кнопку настроек на панели внизу можно перейти в окно настроек, представленное на рисунке 12.

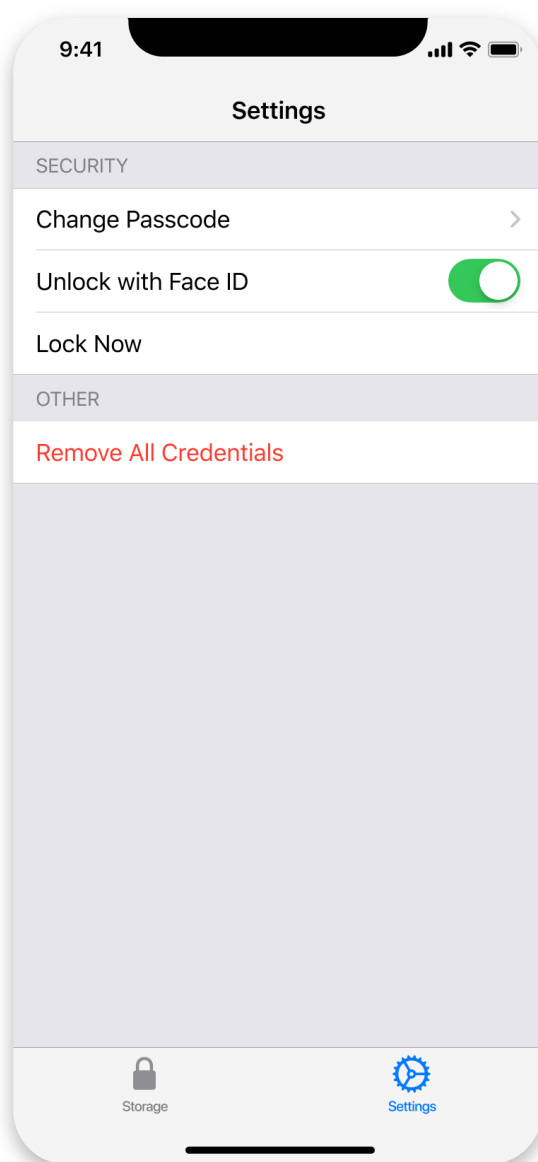


Рисунок 12 – Настройки программы

В меню настроек приложения пользователь может зарегистрировать устройство, синхронизировать данные, передать или принять ключ шифрования и удалить все учётные записи с устройства.

1.6 Технологический стек

Для разработки приложения были использованы следующие технологии:

- объектно-ориентированный подход;
- реляционные базы данных;
- object-relational mapping (ORM);

- кроссплатформенная разработка.

Объектно-ориентированный подход является ключевым аспектом в реализации модульной архитектуры, ORM необходим для простого и безопасного доступа к базе данных [20]. Кроссплатформенная разработка хорошо решает задачу удешевления и ускорения процесса разработки [22].

Исходя из этого был использован следующий стек технологий:

В качестве среды разработки используется IDE «JetBrains Rider». Это удобная среда для разработки приложений на языке «C#».

Для разработки мобильных приложений на ОС «Android» и «iOS» используется платформа «Xamarin» с поддержкой «.NET Standard 2.0».

Для разработки приложения для администрирования сервера используется платформа с открытым исходным кодом «MAUI Blazor». Она позволяет разрабатывать приложения с веб версткой на языке «C#».

Сервер разработан с использованием фреймворка «ASP.NET Core». Для обращения к БД использован «Entity Framework».

Локальное хранилище поддерживается при помощи «SQLite», реализованной на каждой клиентской платформе.

Так как серверное приложение не должно устанавливать стороннее программное обеспечение, то в качестве БД используется «SQLite».

Все общие библиотеки реализованы на «.NET Standard 2.0».

Таким образом вся система была реализована на языке «C#». Макеты интерфейса проектировались с помощью Figma. В качестве системы контроля версий был использован «Git».

Благодаря выбору такой архитектуры и стека технологий описываемая программа имеет следующие качества:

- кроссплатформенность по умолчанию;
- компактность и простота кодовой базы;
- гибкость внесения изменений;
- простой и надёжный способ обращения к базам данных.

Кроссплатформенность по умолчанию сокращает усилия и ресурсы на разработку и поддержку программы, а также предоставляет единую кодовую базу для всех платформ.

Благодаря компактности и простоте реализации, код можно легко прочитать, изменить, отладить. Более простой код также более надёжен.

Благодаря модульной системе можно в любой момент разработки сменить реализацию с минимальными усилиями, например алгоритмы шифрования, реализацию хранилищ и так далее.

ORM облегчает работу с СУБД и предоставляет безопасное по типам обращение [20].

Выводы

В настоящее время большинство менеджеров паролей используют облачную модель хранения и передачи данных. Разработанная модель имеет меньше рисков утечки данных за счёт более децентрализованного развёртывания.

Большинство аналогов в качестве алгоритма шифрования используют AES, а в качестве функции формирования ключей PBKDF2. Это надёжные алгоритмы, проверенные временем, которые имеют давно отлаженную реализацию.

Разработанный проект основывается на готовых решениях для снижения времени разработки и стоимости.

2 Построение моделей и алгоритмов

2.1 Графическая модель хранения и передачи учётных данных без участия устройств третьих лиц

Система для администрирования учётных данных пользователей включают в себя следующие элементы:

- сервер на персональном компьютере;
- приложение для администрирования сервера;
- клиент.

Сервер разворачивается на персональном компьютере и используется для синхронизации учётных данных между устройствами. Для хранения данных и правильного функционирования необходима СУБД.

Приложение для администрирования сервера устанавливается на персональный компьютер. Оно необходимо для управления сервером и регистрации новых устройств.

Клиент устанавливается на пользовательских устройствах, в том числе как расширение для браузеров на персональных компьютерах. Приложение сохраняет учётные данные пользователя и вставляет их в предложенные для авторизации поля.

Диаграмма развёртывания представлена на рисунке 2.

Записи о зарегистрированных устройствах, которым разрешен обмен данными, хранятся в сервисе.

Подобные записи содержат уникальные идентификаторы, пользовательские названия, модели устройств и даты последней синхронизации.

Сохранённые учётные данные необходимы для их вставки в предоставляемые сайтами и приложениями формы для авторизации.

Записи этих данных содержат идентификатор ресурса, логин и пароль в зашифрованном виде, время создания и последней.

2.2 Графическое представление алгоритма синхронизации учётных данных с беспарольной защитой

Синхронизация выполняется по следующему алгоритму: устройство делает выборку учётных данных, которые он не отправлял. Далее создаётся запрос, который содержит ключ аутентификации, дату последней синхронизации и выборку, созданную ранее.

В зависимости от ключа аутентификации сервер может отказать в обработке или принять запрос. В последнем случае сервер отправляет учётные данные, которых нет на устройстве, запрашивающем синхронизацию.

Передача данных при синхронизации может проходить как в локальной, так и в сети интернет. При передаче данных есть риск что их может перехватить «человек посередине». В этом случае данные попадут к третьим лицам. Существует несколько вариантов решения этой проблемы.

Первый вариант – передавать данные без шифрования «как есть» через протокол HTTP. В таком случае в открытом виде злоумышленник получит информацию об используемых ресурсах. Конфиденциальные данные такие как имя пользователя и пароль шифруются и дешифруются непосредственно при использовании приложения мастер-ключом.

Второй вариант – шифровать всю информацию и передавать её через протокол HTTP. В этом случае злоумышленник получит шифротекст, для которого требуется ключ. После дешифровки шифротекста злоумышленник получит информацию из первого варианта. Можно использовать разные ключи для шифрования передаваемых сообщений и учётных данных. Чтобы ещё сильнее повысить безопасность мастер-ключа, можно генерировать временные ключи, используемые для передачи данных. Однако существует проблема передачи ключей шифрования.

Третий вариант – использовать протокол HTTPS, который создан специально, чтобы предотвратить подобную атаку. Такой протокол решит проблему с идентификацией и надёжной передачей данных. Недостатком этого

решения является необходимость регистрировать и устанавливать сертификат. Именно такой способ используется в существующих облачных решениях.

Графическое представление существующего алгоритма синхронизации данных представлена на рисунке 13.

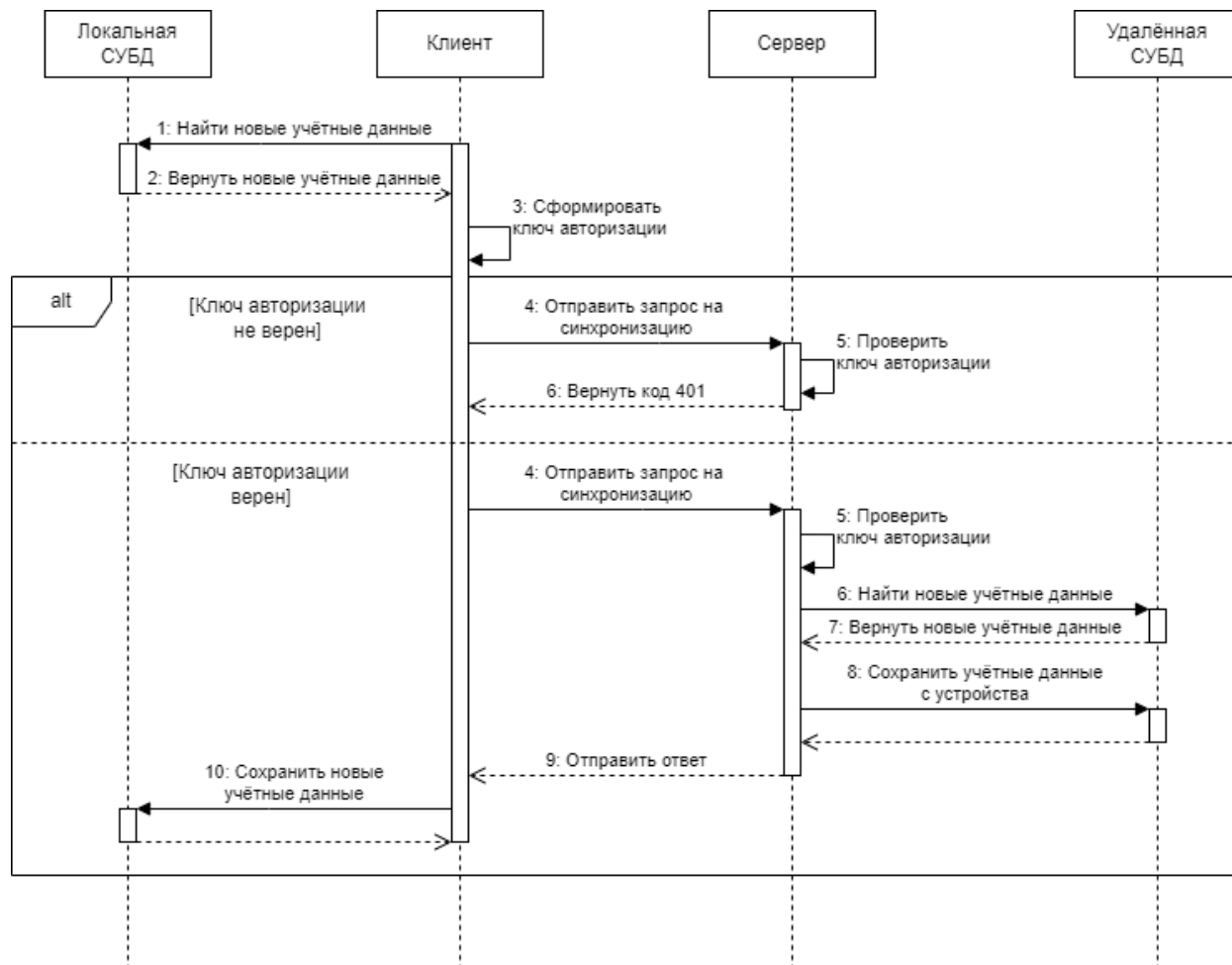


Рисунок 13 – Графическое представление существующего алгоритма синхронизации данных

Установка стороннего сертификата выполнима не на всех устройствах. А регистрация стандартного сертификата требует участия третьих лиц, что противоречит принципу работы проекта.

Таким образом, наиболее подходящий вариант – второй.

Главное отличие разработанного алгоритма от существующего заключается в том, что на этапе синхронизации осуществляется дополнительная проверка на наличие устройства в списке зарегистрированных.

Ключ шифрования является временным и создаётся на основе мастер-ключа и временной метки в пределах 60 секунд как на стороне клиента, так и на стороне сервера. Ключ аутентификации создаётся на основе ключа шифрования с использованием уникального идентификатора устройства.

Получив запрос, сервер проверяет ключ аутентификации. При удачной аутентификации сервер сохраняет изменения, делает выборку для устройства и отправляет её с кодом 200. В противном случае отправляется код 401.

Графическое представление разработанного алгоритма представлена на рисунке 14. Исходный код алгоритма представлен в приложении Г.

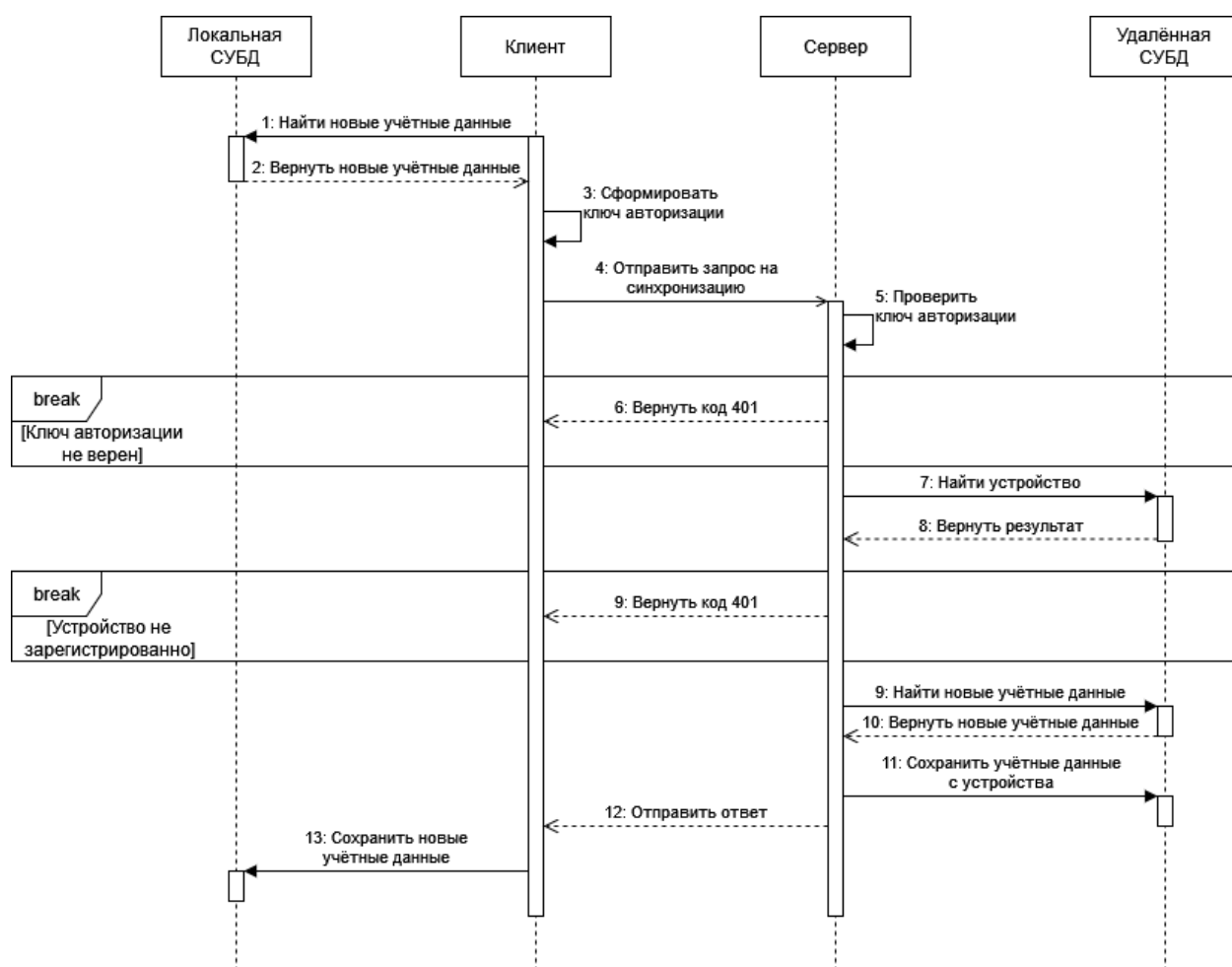


Рисунок 14 – Графическое представление разработанного алгоритма синхронизации данных

Рассмотренные модели и алгоритмы обеспечивают требуемый функционал и безопасность, а также предоставляют свободу выбора реализации средств хранения и шифрования данных.

2.3 Графическое представление алгоритма регистрации устройства и передачи ключа

Перед использованием приложения, необходимо установить и настроить серверное приложение. При первом запуске сервер генерирует случайный мастер-ключ, который шифруется средствами ОС и сохраняется. Этот мастер-ключ будет использоваться для шифрования учётных данных.

Далее устройства пользователя должны пройти процесс регистрации на сервере. Графическое представление разработанного алгоритма представлена на рисунке 15.

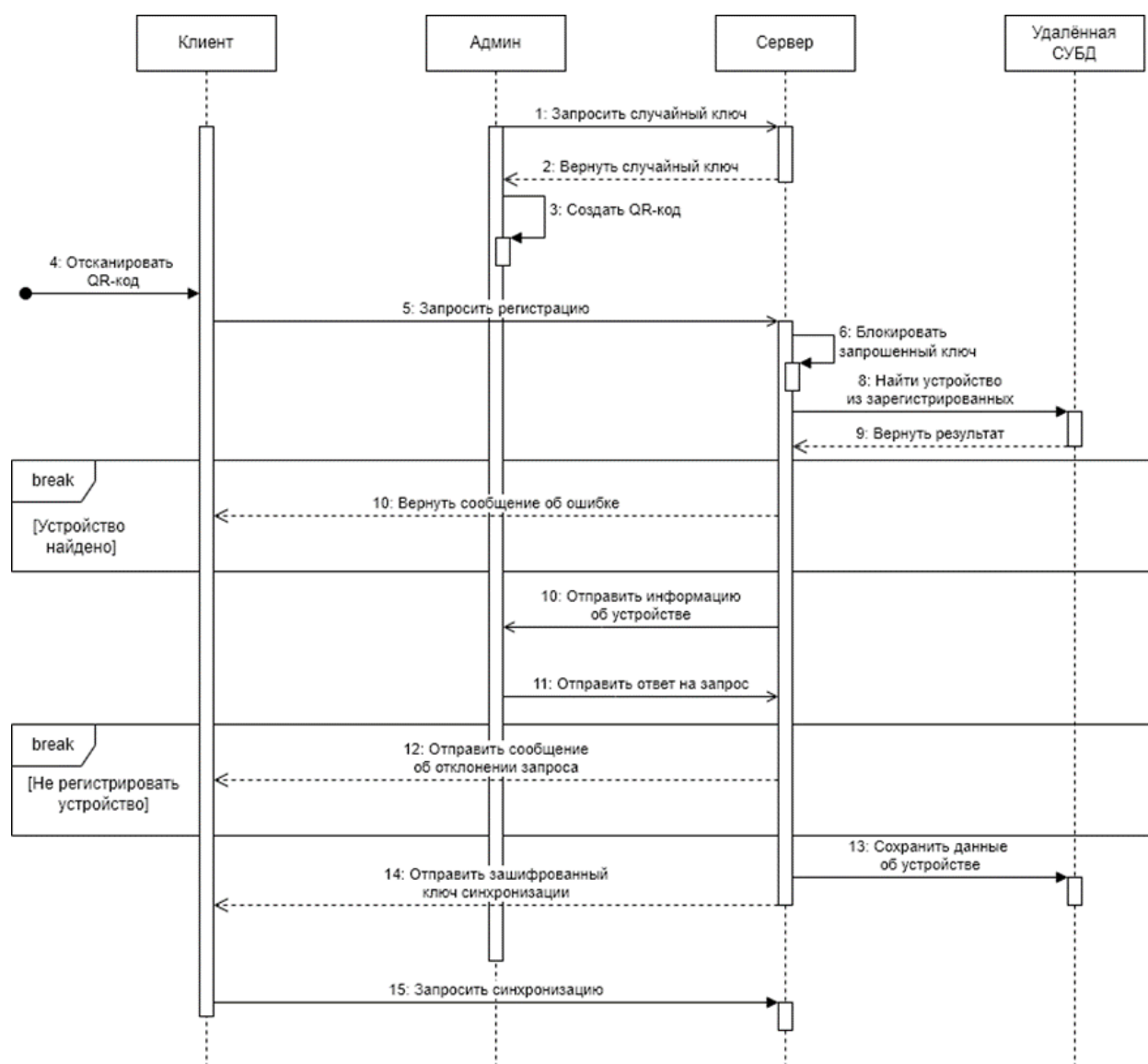


Рисунок 15 – Графическое представление разработанного алгоритма регистрации и передачи ключа

При регистрации устройства приложение на сервере генерирует QR-код, который необходимо отсканировать на устройстве. В нём содержится информация, необходимая для доступа к серверу по локальной сети (случайный ключ шифрования). По нему можно зарегистрировать не более одного устройства, после чего он становится недействительным.

После сканирования QR-кода, устройство отправляет запрос на регистрацию, содержащий информацию о себе. Для продолжения регистрации пользователь должен проверить и подтвердить запрос на сервере. После этого сервер сохраняет информацию об устройстве и отправляет на него мастер-ключ.

Такой способ регистрации был выбран потому, что передача ключей по нему не подвержена атаке посредника, так как для её осуществления злоумышленнику необходимо иметь физический доступ к серверу. Исходный код алгоритма представлен в приложении В.

Выводы

Разработана графическая модель хранения и синхронизации, отличающийся от существующих отсутствием в системе устройств третьих лиц, позволяющая хранить и передавать данные более децентрализованно, а также разработан алгоритм синхронизации данных, отличающийся от существующих беспарольной системой защиты, и не позволяющий третьим лицам получить несанкционированный доступ к паролю пользователя.

3 Реализация моделей и алгоритмов

3.1 Программная реализация модели и алгоритмов синхронизации

Разрабатываемая система состоит из четырёх наборов проектов: «Shared», «Client», «Admin», «Server». Её структура представлена на рисунке 16.

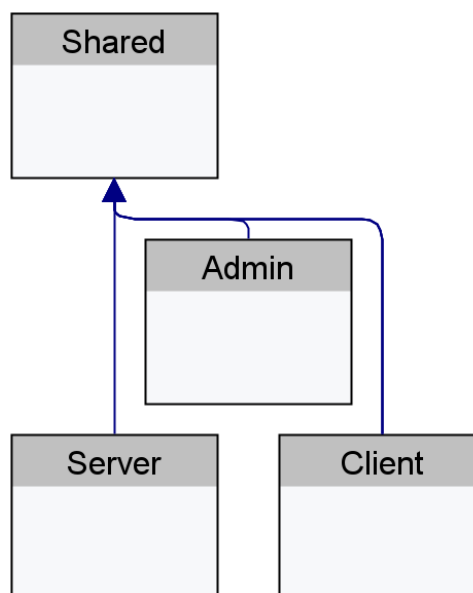


Рисунок 16 – Схема наборов библиотек

«Shared», представленный на рисунке 17 – это набор проектов, реализующий общие классы и методы, которые используются в других модулях. Он состоит из трех подмодулей: библиотеку «Credentials», которая отвечает за представление учётных данных, библиотеку «Crypto», которая описывает абстракцию над шифрованием данных и библиотеку «Communication», которая реализует методы, используемые для передачи данных между проектами на разных устройствах.

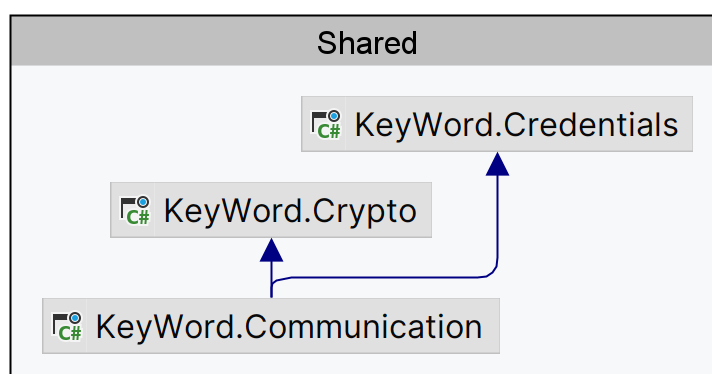


Рисунок 17 – Схема модулей общей библиотеки

Библиотека «Credentials» отвечает за представление учётных данных. Её классы представлены на рисунке 18.

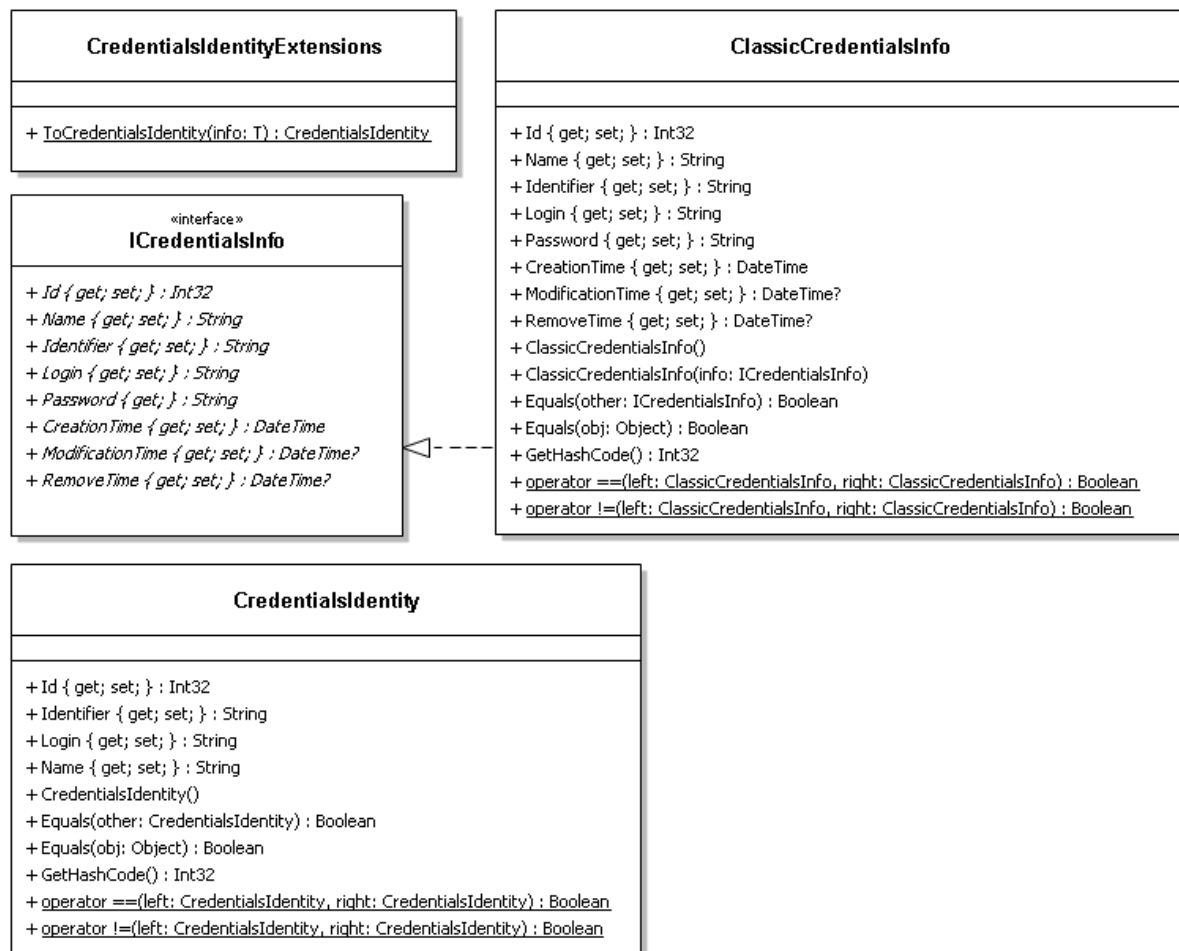


Рисунок 18 –Диаграмма классов проекта «Credentials»

Интерфейс «ICredentialsInfo» описывает функционал типичной учётной записи. В нём описаны следующие свойства:

- «Id» – идентификатор учётной записи, выраженный целым числом;
- «Name» – имя учётной записи, которую видит пользователь в интерфейсе;
- «Login» – имя пользователя в учётной записи;
- «Password» – пароль учётной записи;
- «CreationTime» – время создания учётной записи;
- «ModificationTime» – время последнего изменения учётной записи;
- «RemoveTime» – время удаления учётной записи.

При удалении учётной записи стирается вся информация в базе данных и сохраняется время удаления. Это необходимо для обеспечения синхронизации учётных данных.

«ClassicCredentialsInfo» является реализацией интерфейса «ICredentialsInfo». Основной идеей этого класса является сохранение пароля как последовательности символа. Альтернативой такому подходу может быть процедурная генерация ключа на основе идентификационных данных учётной записи (идентификатора, url-ресурса, имени пользователя), вместо хранения в базе данных.

Класс «CredentialsIdentity» реализует учётную сущность, которая служит для описания и идентификации учётной записи. За идентификацию учётной записи отвечают следующие поля: Id, Identifier, Login, Name.

Статический класс «CredentialsIdentityExtensions» реализует единственный метод расширения «ToCredentialsIdentity», который преобразует учётную запись в учётную сущность.

Библиотека «Crypto» предоставляет абстракцию методов шифрования и криптографии. Его классы представлены на рисунке 19.

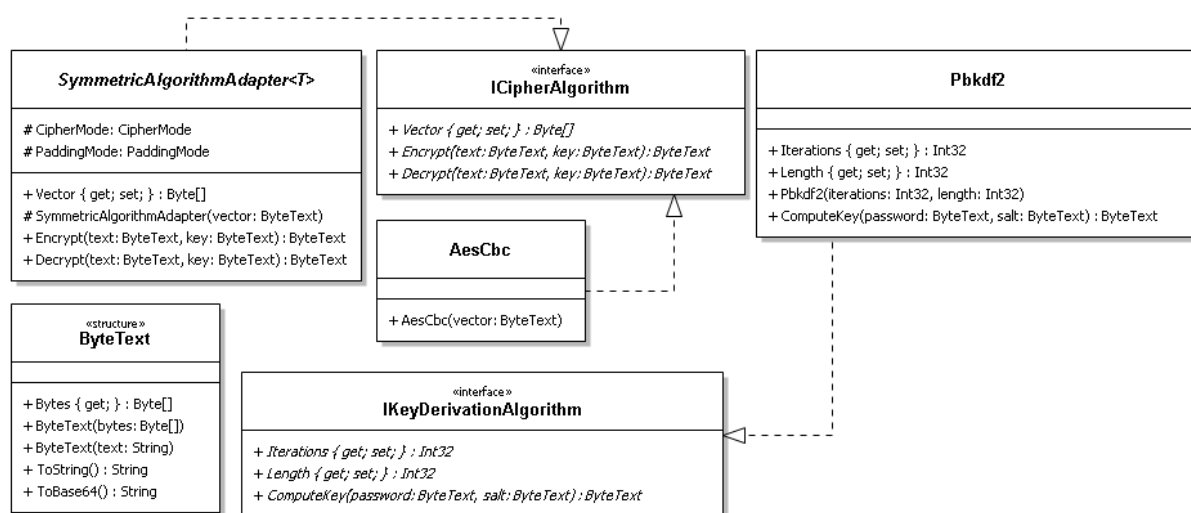


Рисунок 19 – Диаграмма классов библиотеки «Crypto»

Структура «ByteText» хранит текст в виде байтов и предоставляет удобные способы преобразования байтов в текст и наоборот.

Интерфейс «IKeyDerivationAlgorithm» описывает абстрактный генератор ключа. Свойство «Iterations» хранит количество итераций функции. Свойство «Length» хранит длину генерируемого ключа. Метод «ComputeKey» генерирует ключ на основе пароля и криптографической соли.

Класс «Pbkdf2» реализует интерфейс «IKeyDerivationAlgorithm» и функцию формирования ключа Pbkdf2.

Интерфейс «ICipherAlgorithm» описывает типичный алгоритм шифрования. Он может быть как симметричным, так и асимметричным. Свойство «Vector» задаёт вектор инициализации, который используется в блочных алгоритмах шифрования. Метод «Encrypt» принимает исходный текст и ключ и возвращает зашифрованный текст. Метод «Decrypt» принимает зашифрованный текст и ключ, возвращает исходный текст, являясь обратной предыдущей функции.

Класс «SymmetricAlgorithmAdapter» – это абстрактный класс, который необходим для адаптации встроенных реализаций под интерфейс «ICipherAlgorithm».

Класс «AesCbc» реализует метод шифрования AesCbc. Создан на основе предыдущего класса и адаптирует внутренний класс «AesManaged».

Библиотека «Communication» реализует методы и константы, используемые для передачи данных между проектами на разных устройствах. Его классы представлены на рисунке 20.

Класс «SyncUtilities» реализует набор функций для коммуникации клиента и сервера при синхронизации. Функция «GetDeviceAuthKey» возвращает временный ключ аутентификации, который используется при обмене учётными данными. Функции «GetDiscoveryRequestAuthKey» и «GetDiscoveryResponseAuthKey» возвращают временные ключи запроса и ответа, когда в процессе регистрации устройство ищет сервер в локальной сети. Функция «GetAuthId» возвращает идентификатор, который используется для разграничения данных, хранимых на сервере.

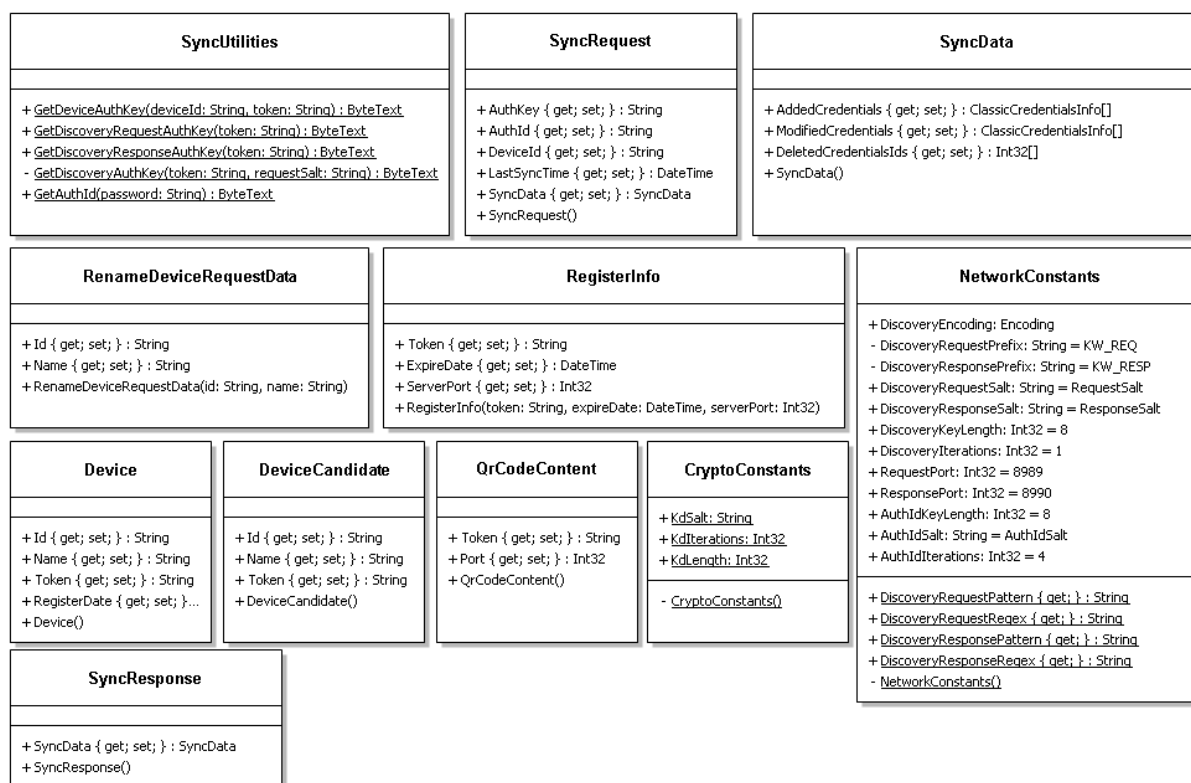


Рисунок 20 – Диаграмма классов проекта «Communication»

Класс «SyncRequest» – реализует форму запроса на синхронизацию. Содержит следующие поля:

- «AuthKey» –ключ аутентификации;
- «AuthId» – идентификационный ключ, необходимый для разграничения данных. По сути является случайным значением;
- «DeviceId» – идентификатор устройства, зарегистрированного на сервере;
- «LastSyncTime» – временная метка последней синхронизации. Необходимо для оптимизации объёма данных при ответе;
- «SyncData» – данные синхронизации.

Класс «SyncResponse» – реализует форму ответа на запрос синхронизации. Хранит в себе данных синхронизации.

Объекта класса «SyncData» хранят в себе данные, необходимые для синхронизации. Это добавленные и изменённые учётные данные, а также идентификаторы тех данных, которые были удалены.

Объекты класса «RegisterInfo» хранят информацию текущей регистрационной сессии. Содержит в себе следующие поля:

- «Token» необходим для защиты от подбора. Он же будет присвоен устройству и использоваться при шифровании;
- «ExpireDate» – дата и время устаревания сессии. При наступлении этого срока сессия становится недействительной и создаётся новая;
- «ServerPort» – TCP порт для обмена информации. Может браться случайно.

Класс «Device» необходим для описания зарегистрированного устройства. В нём есть идентификатор, название устройства, которое будет видеть пользователь в интерфейсе, токен, при помощи которого будет осуществляться аутентификация и дата регистрации.

Объекты класса «DeviceCandidate» хранят ту же информацию, что и объекты класса «Device», но без даты регистрации. Используется при подтверждении регистрации устройства.

Объект класса «RenameDeviceRequestData» используется для запроса смены названия устройства.

В объекте класса «QrCodeContent» хранятся данные, которые преобразуются в qr-код, необходимый для считывания устройством в процессе регистрации. В qr-код кодируются токен и порт.

Статический класс «CryptoConstants» хранит в себе константы, которые используются при шифровании и дешифровании данных.

Статический класс «NetworkConstants» хранит константы, необходимые для корректного обмена данными между клиентом, сервером и приложением для администрирования.

Набор проектов «Server», представленный на рисунке 21, состоит из двух частей: библиотеки-интерфейса базы данных и исполняемого проекта. Такое разделение позволит использовать любую реализацию базы данных.

Библиотека «Server Storage», представленная на рисунке 22, объявляет абстракции, а также стандартную реализацию для работы сервера с хранилищем данных.

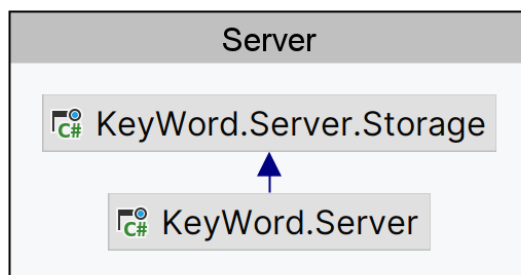


Рисунок 21 – Схема проектов сервера

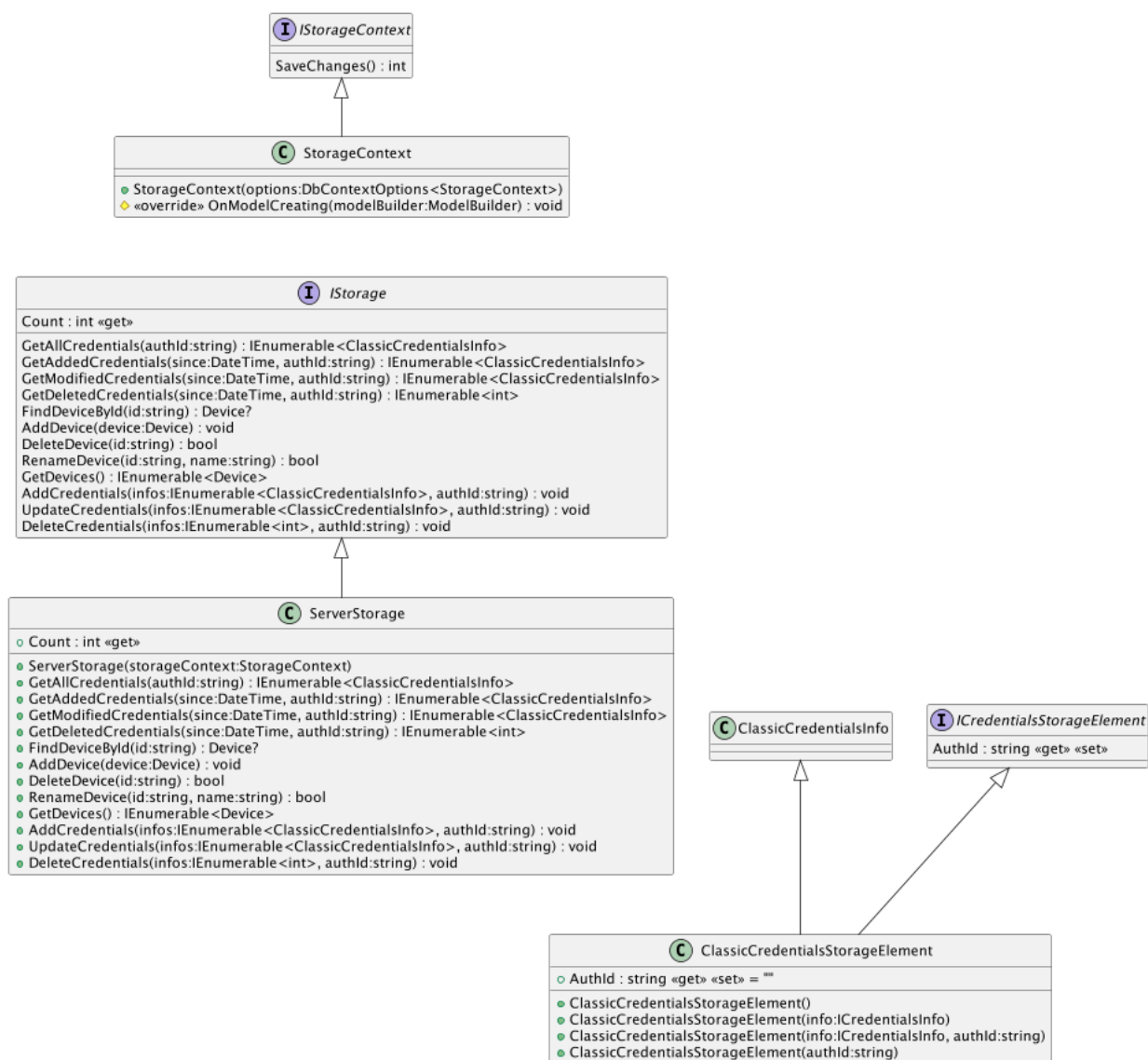


Рисунок 22 – Диаграмма классов библиотеки «Server Storage»

Интерфейс «IStorage» описывает необходимые методы и свойства для работы сервера с хранилищем данных. Благодаря выделению такой абстракции, может быть использована реализация на основе любой базы данных.

Интерфейс «IStorageContext» описывает необходимые методы и свойства для реализации хранилища, использующего «Entity Framework».

Классы «ServerStorage», «StorageContext» и «ClassicCredentialsStorageElement» являются стандартной реализацией хранилища данных с использованием СУБД «SQLite». Схема базы данных представлена на рисунке 23.

На схеме представлены две сущности: «CredentialsStorageElement» и «Device».

Сущность «CredentialsStorageElement» необходима для хранения зашифрованной учётной записи с устройства. Поле «AuthId» хранит отпечаток ключа шифрования. Таким образом, учётные данные с разных пользователей могут храниться одновременно в одном хранилище.

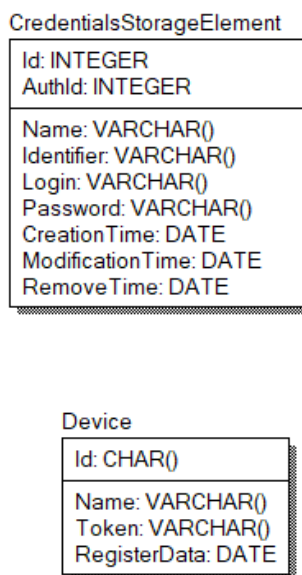


Рисунок 23 – Диаграмма IDEF1X базы данных «Server Storage»

Сущность «Device» необходима для хранения зарегистрированных устройств, для которых разрешена передача данных. Для них сохраняется название, токен и дата регистрации.

Проект «Server» представляет собой серверное приложение, реализованное с помощью «ASP.NET». В проекте представлено API, которое реализует функции регистрации, синхронизации и управлением зарегистрированными устройствами. Эти функции соответственно выполняют три контроллера: «RegisterController», «SyncController» и «DevicesController». Их схема представлена на рисунке 24. Все удалённые вызовы сервера реализованы через протокол HTTP.



Рисунок 24 – Диаграмма классов проекта «Server»

Контроллер «RegisterController» отвечает за процесс регистрации устройства. В нём доступны следующие вызовы:

- «Register/StartNewRegistration»;
- «Register/RequestNewToken»;
- «Register/RequestDeviceCandidate»;
- «Register/PostDeviceInfo»;
- «Register/GetDeviceApproval»;
- «Register/ApprovePendingDevice»;
- «Register/DenyPendingDevice».

Вызов «Register/StartNewRegistration» доступен только с приложения для администрирования. Создаётся новая сессия «RegisterSession», с ограниченным временем жизни и случайно сгенерированным токеном. Сессия закрывается по истечению времени жизни или если создана новая сессия.

Вызов «Register/RequestNewToken» доступен только с приложения для администрирования. Он возвращает токен из текущей сессии. Если таковой нет, то будет возвращён код «404».

Вызов «Register/RequestDeviceCandidate» доступен только с приложения для администрирования. Это долгий вызов. Он ждёт, когда на сервер будет отправлен кандидат зарегистрированного устройства. Когда информация об устройстве попадает на сервер или она уже есть, этот вызов сразу возвращает её. Если регистрационная сессия закрывается, возвращается код «503». Если вызов происходит без активной сессии, возвращается код «400».

Вызов «Register/PostDeviceInfo» происходит с устройства для регистрации. В нём содержится информация об устройстве. При вызове делается проверка на существовании сессии, валидности токена и занятости сессии. При провале одной из проверок, отправляется код «400». Если всё проходит успешно, сессия запоминает устройство и становится занятой. Больше сервер не принимает никаких новых устройств.

Вызов «Register/GetDeviceApproval» происходит с устройства для регистрации. Это долгий вызов, который ожидает решения пользователя. Если

устройство успешно зарегистрировано или уже есть в базе данных, возвращается код «200». В противном случае возвращается код «403».

Вызовы «Register/ApprovePendingDevice» и «Register/DenyPendingDevice» доступны только с приложения для администрирования. Они характеризуют разрешение пользователя на подтверждение или отклонение регистрации устройства. После одного из этих вызовов сессия сразу закрывается. Оба вызова могут вернуть код «400», если сессии или устройства не существует или жизненное время сессии окончено.

Контроллер «SyncController» отвечает за синхронизацию данных. В нём доступен единственный вызов «Sync/RequestSync». Он получает запрос синхронизации от устройства. Перед началом синхронизации проверяется регистрация устройства, его токен и отпечаток ключа шифрования. Если провале одной из проверок отправляется код «401». Далее сервер выбирает данные, которые необходимы с момента последней регистрации и отправляются. Данные с устройства сохраняются в хранилище.

Контроллер «DevicesController» отвечает за управление зарегистрированными устройствами. В нём доступны следующие вызовы:

- «Devices/GetDeviceList»;
- «Devices/DeleteDevice»;
- «Devices/RenameDevice».

Все вызовы доступны только с приложения для администрирования.

Вызов «Devices/GetDeviceList» возвращает список всех зарегистрированных устройств.

Вызов «Devices/DeleteDevice» удаляет зарегистрированное устройство, если такое существует. Если нет, возвращается код «404».

Вызов «Devices/RenameDevice» меняет название зарегистрированного устройства, если такое существует. Если нет, возвращается код «404».

Набор проектов «Admin», представленный на рисунке 25, состоит из двух частей: библиотеки с набором сервисов, необходимых для коммуникации с

сервером и исполняемого проекта. Такое разделение позволит отлаживать вызовы в отдельность от пользовательского интерфейса.

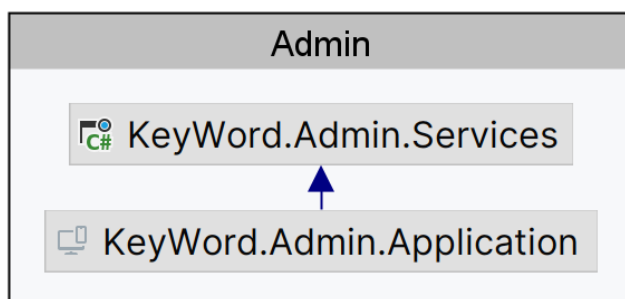


Рисунок 25 – Схема проектов сервера

Проект «Admin.Services», показанный на рисунке 26, собирается в библиотеку, которая реализует HTTP запросы к серверу. Она используется в основном приложении.

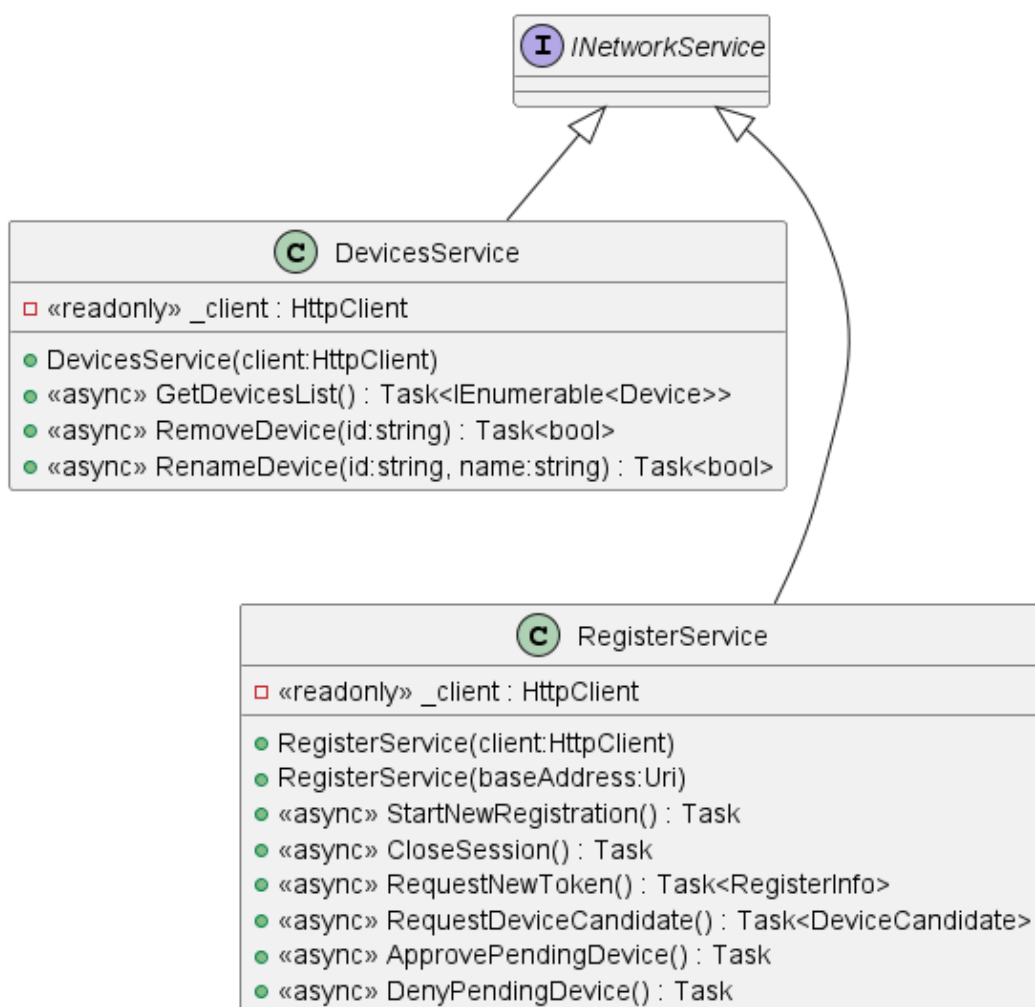


Рисунок 26 – Диаграмма классов проекта «Admin.Services»

Проект «Admin.Application» является приложением, реализованным с помощью «MAUI Blazor». На данный момент приложение содержит две страницы: страницу приветствия и страницу списка зарегистрированных устройств. В последней можно добавлять, удалять и переименовывать зарегистрированные устройства. Страница списка устройств и процедура регистрации показаны на рисунках 27 и 28.

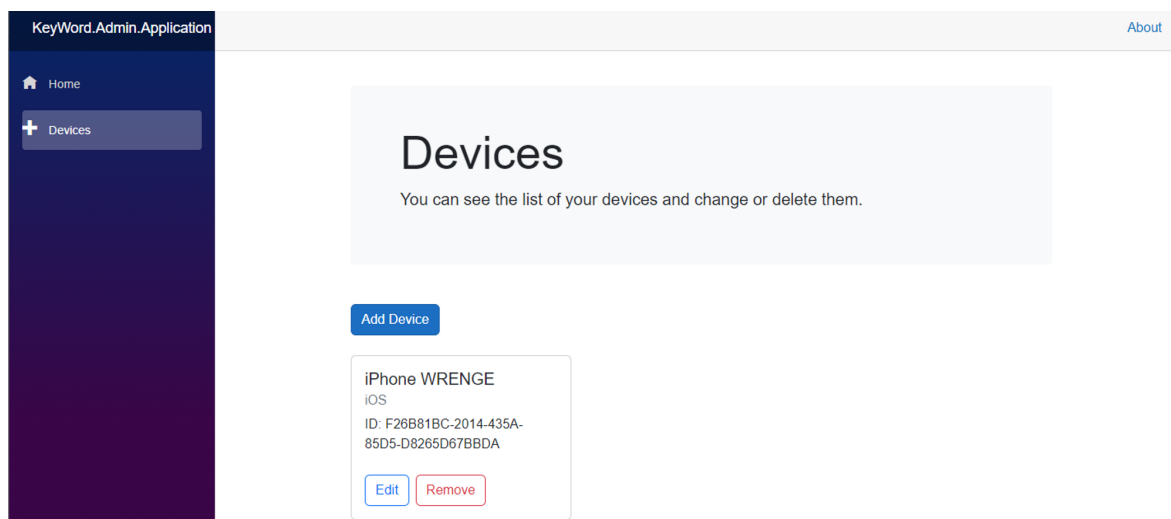


Рисунок 27 – Страница списка устройств приложения «Admin.Application»

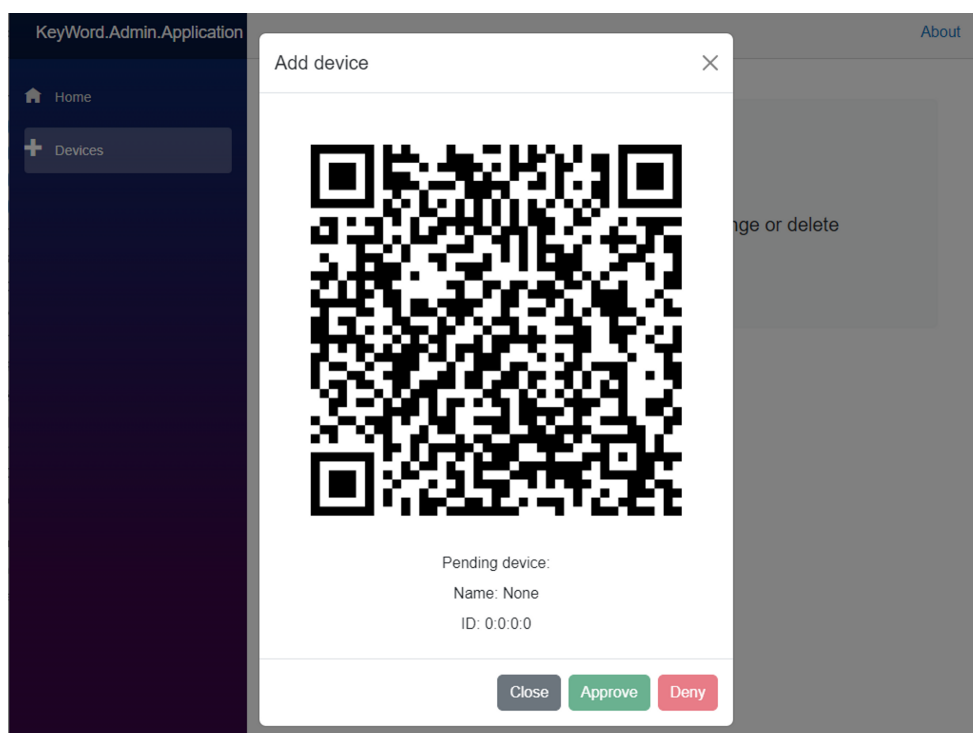


Рисунок 28 – Процедура регистрации нового устройства приложения «Admin.Application»

В набор «Client», представленный на рисунке 29, входит шесть проектов: два проекта приложения под каждую платформу и общая библиотека, проект, описывающий абстракцию хранилища, проект с реализацией хранилища и проект с реализацией вызовов к серверу.

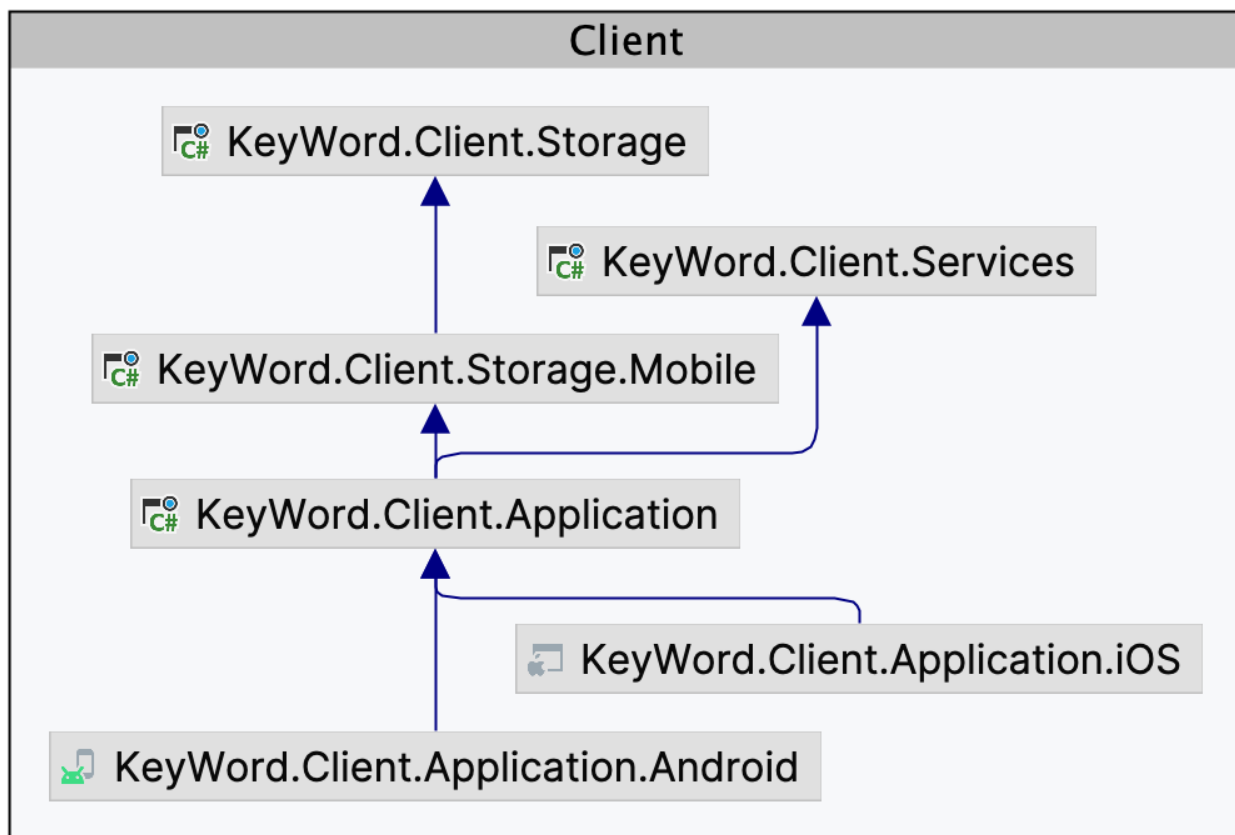


Рисунок 29 – Схема проектов клиентского приложения

В проекте «Client.Services», показанном на рисунке 30, реализованы классы с методами для осуществления HTTP запросов к серверу. Такое отделение приложения и запросов имеет несколько преимуществ. Во-первых, функционал, отделённый от интерфейса проще поддерживать и изменять. Во-вторых, такую реализацию можно автоматически тестировать в интеграционных тестах.

В проекте «Client.Storage» описаны интерфейс взаимодействия с хранилищем и исключения, которые могут возникнуть. Диаграмма классов этого проекта представлена на рисунке 31.

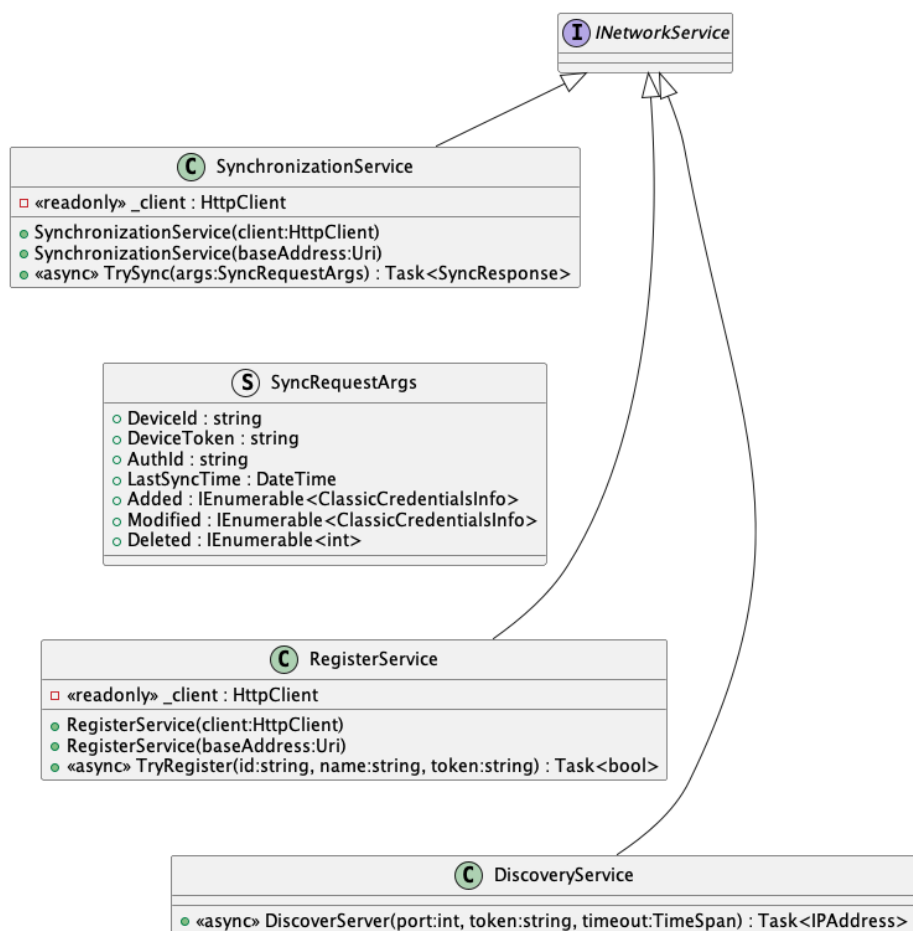


Рисунок 30 – Диаграмма классов проекта «Client.Services»

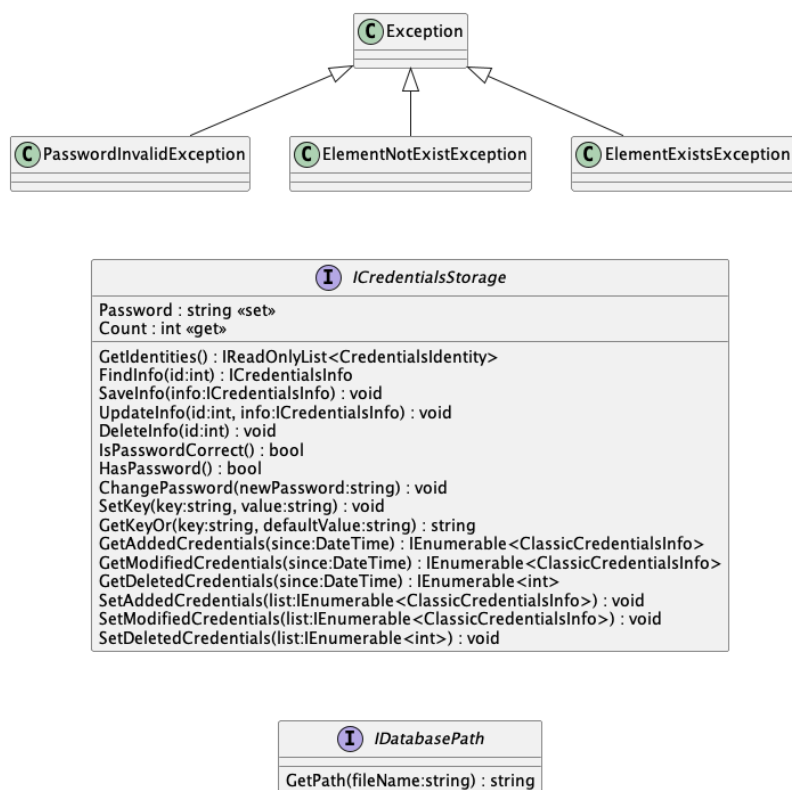


Рисунок 31 – Диаграмма классов проекта «Client.Storage»

В проекте «Client.Storage.Mobile» реализовано само хранилище через СУБД SQLite и EntityFramework. Такое разделение позволяет использовать разные реализации хранилищ через разные СУБД. Диаграмма классов представлена на рисунке 32 соответственно.



Рисунок 32 – Диаграмма классов проекта «Client.Storage.Mobile»

В интерфейсе «ICredentialsStorage» описаны следующие свойства и методы:»

- «Password» – пароль, введенный пользователем;
- «Count» – количество учетных записей в хранилище;
- «GetIdentities» – получить все учетные сущности;
- «FindInfo» – найти учетную запись по её id.
- «SaveInfo» – сохранить новую учетную запись;
- «UpdateInfo» – обновить имеющуюся учетную запись;
- «DeleteInfo» – удалить учетную запись;
- «IsPasswordCorrect» – проверить, корректен ли пароль;
- «HasPassword» – проверить, задан ли пароль у хранилища;
- «ChangePassword» – сменить пароль (работает при правильно введенном пароле, либо если пароль не задан);
- «SetKey» – задать значение ключу;
- «GetKeyOr» – получить значение по ключу;
- «GetAddedCredentials» – найти добавленные учетные записи с момента времени;
- «GetModifiedCredentials» – найти измененные учетные записи с момента времени;
- «GetDeletedCredentials» – найти удаленные учетные записи с момента времени;
- «SetAddedCredentials» – добавить учетные записи извне;
- «SetModifiedCredentials» – обновить учетные записи извне;
- «SetDeletedCredentials» – удалить учетные записи извне.

Кроме хранилища учетных записей, существует хранилище по типу ключ-значение. В нём хранится дата последней синхронизации и отпечаток ключа шифрования.

Отпечаток ключа шифрования необходим для проверки корректности введенного ключа.

Все учётные записи в хранилище шифруются ключом шифрования. Он задаётся через свойство «Password».

Методы «SetAddedCredentials», «SetModifiedCredentials» и «SetDeletedCredentials» нужны для обновления хранилища при синхронизации.

Классы, реализующие интерфейс «IDatabasePath» необходимы для того, чтобы найти файл базы данных на какой-либо платформе.

В проекте «Client.Storage» описаны следующие исключения:

- «PasswordInvalidException» – пароль некорректен (выбрасывается во всех методах, где имеет место чтение и запись данных в хранилище);
- «ElementNotExistException» – учётная запись не найдена (выбрасывается в «UpdateInfo» и «DeleteInfo»);
- «ElementExistsException» – учётная запись уже существует (выбрасывается в «SaveInfo»).

Проект «Client.Application» является приложением, реализованным с помощью «Xamarin». Оно собирается под две платформы: «Android» и «iOS». Под каждую платформу существует свой проект с деталями реализации: «Client.Applicaion.Android» и «Client.Application.iOS». В приложении разработано пять окон:

- окно списка учётных записей;
- окно описания учётной записи;
- окно добавления/редактирования учётной записи;
- окно настроек, окно регистрации.

Окна приложения представлены на рисунках 33 - 36.

Для корректной работы приложения были необходимы следующие требования: разрешение на использование камеры, которое необходимо для сканирования qr-кода в процессе регистрации, и разрешение на сетевое обнаружение, которое необходимо, чтобы найти сервер в локальной сети и отправить регистрационные данные.

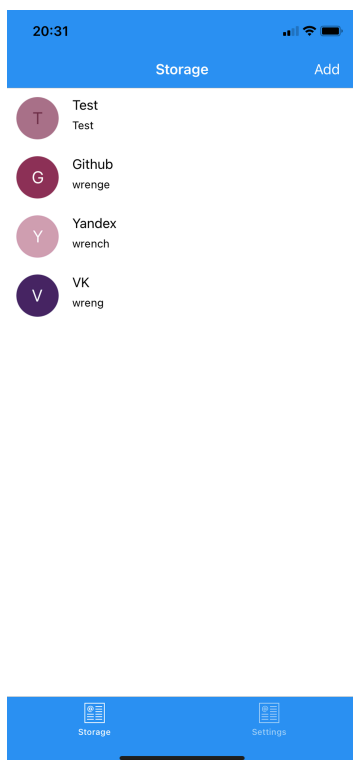


Рисунок 33 – Окно списка учётных записей

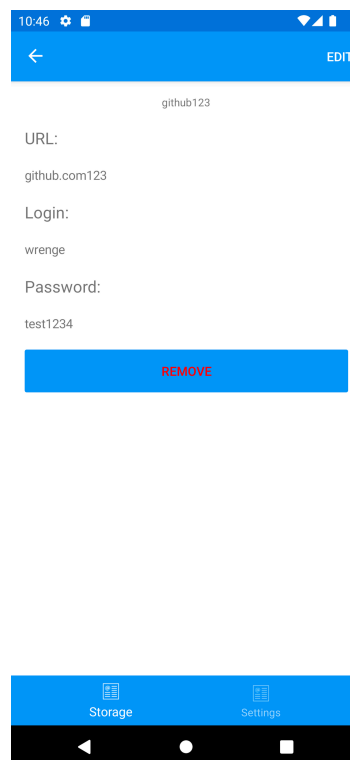


Рисунок 34 – Окно описания учётной записи

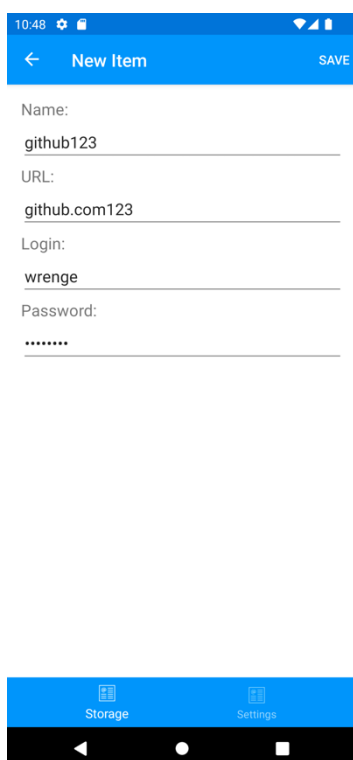


Рисунок 35 – Окно добавления/редактирования учётной записи

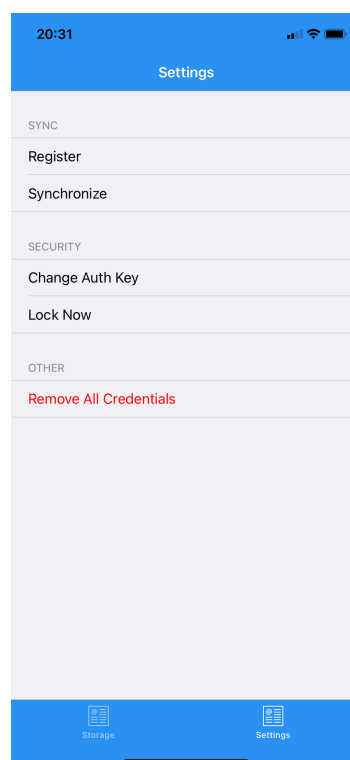


Рисунок 36 – Окно настроек приложения

Также были использованы методы, специфичные для каждой платформы. В частности, классы «AndroidDatabasePath» и «IosDatabasePath» необходимые для того, чтобы получить путь до файла-хранилища, а класс «DeviceUidService» необходим для получения уникального кода, идентифицирующего устройство. Оно необходимо при регистрации.

Полный исходный код программы можно посмотреть по ссылке в github [23].

3.2 Тестирование реализации модели и алгоритмов синхронизации

Применение метрик позволяет руководителям проектов и предприятий изучить сложность разработанного или даже разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения. Однако метрики могут служить лишь рекомендательными характеристиками, ими нельзя полностью руководствоваться, так как при разработке ПО программисты, стремясь минимизировать или максимизировать ту или иную меру для своей программы, могут прибегать к хитростям вплоть до снижения эффективности работы программы. В более сложной программе ошибку найти сложнее.

Для анализа программного кода были использованы IDE Visual Studio и следующие метрики:

- индекс удобства поддержки (MI);
- цикломатическая сложность (CyC);
- глубина наследования (DoI);
- взаимозависимость классов (ClCo);
- количество строк кода (LoS);
- количество строк исполняемого кода (LoE).

Индекс удобства поддержки (Maintainability Index) – комплексный показатель качества кода. Этот показатель разработан специалистами из Carnegie Mellon Software Engineering Institute. Рассчитывается метрика по следующей формуле:

$$MI = \max\left(0, (171 - 5.2 \cdot \ln(HV) - 0.23 \cdot CC - 16.2 \cdot \ln(LoC)) \cdot \frac{100}{171}\right)$$

HV – Halstead Volume, вычислительная сложность. Чем больше операторов, тем больше значение этой метрики;

CC – Cyclomatic Complexity. Эта метрика описана ниже;

LoC – количество строк кода.

Эта метрика может принимать значения от 0 до 100 и показывает относительную сложность поддержки кода. Чем больше значение этой метрики, тем легче поддерживать код.

Сам Visual Studio выделяет три уровня:

- 1) зелёный – от 20 до 100 означает хороший код;
- 2) жёлтый – от 10 до 19 означает средний код;
- 3) красный – от 0 до 9 означает, что код лучше переписать.

Эти группы и являются субъективными и у каждого разработчика есть своя оценка по этой шкале.

Цикломатическая сложность (Cyclomatic Complexity) показывает структурную сложность кода, то есть количество различных ветвей в коде. Чем больше этот показатель, тем больше тестов должно быть написано, для полного покрытия кода.

Глубина наследования (Depth of Inheritance) показывает для каждого класса, какой он по счету в цепочке наследования. Например, есть 3 класса А, В, С, В унаследован от А, а С унаследован от В, то значение этой метрики для классов А, В и С будет равно соответственно 1, 2 и 3.

Взаимозависимость классов (Class Coupling) показывает степень зависимости классов друг с другом. В расчет берутся уникальные классы из параметров, локальных переменных, возвращаемого типа, базового класса, атрибутов. Хороший дизайн программного обеспечения предполагает небольшое количество связанных классов. Чем их больше, тем сложнее в дальнейшем переиспользовать этот класс, а также поддерживать, так как существует очень много зависимостей.

Количество строк исполняемого кода (Lines of Code) показывает количество строк кода. Этот показатель показывает не точное количество строк в вашем файле, так как подсчет основан на IL-коде. В расчет не берутся пустые строчки, комментарии, строчки со скобками, объявление типов и пространств имен. Большое количество строк в методе/классе может показывать на ошибки в проектировании и на то, что этот код можно разделить на несколько частей.

Результаты расчётов метрик проектов представлены в таблице 4.

Таблица 4 – Метрики исходного кода

Project	MI	CyC	DoI	ClCo	LoS	LoE
Server.Tests	69	27	1	54	664	309
IntegrationTests	70	25	2	55	415	141
Server	72	85	2	94	381	223
Client.Services	74	11	1	33	145	47
Credentials.Test	79	3	1	5	64	14
Admin.Application	82	70	8	77	1623	184
Client.Storage.Mobile	82	81	2	47	474	157
Client.Storage.Mobile.Tests	82	14	1	29	263	96
Client.Application	83	206	8	97	1206	382
Admin.Services	83	24	1	21	123	38
Crypto.Tests	83	2	1	6	25	13
Credentials	86	73	1	7	144	44
Client.Application.iOS	88	5	4	17	57	11
Crypto	90	33	2	18	142	37
Client.Application.Android	91	41	11	103	35365	16415
Server.Storage	91	54	2	28	203	72
Communication	92	61	1	14	158	49
Client.Storage	100	22	2	7	53	2

Из представленной таблицы можно сделать вывод, что ни один из проектов не попадает в «красную зону». Самые низкие показатели в основном у проектов для модульных тестов из-за того, что в них сконцентрированы вызовы всех

функций тестируемого проекта по несколько раз, из-за чего повышается вычислительная сложность.

Так же анализатор сканирует сгенерированный код, качество которого обычно значительно хуже. Генерация кода используется во всех проектах, кроме «Credentials», «Crypto», «Communication». Самый яркий пример – проект «Client.Application.Android» с его колоссальным количеством строк кода. Реальный же показатель – 36 строк исходного кода.

У проекта с самой большой цикломатической сложностью «Client.Application» высокий показатель является суммарным с каждого класса, в том числе и сгенерированных. Ни у одного класса показатель не превышает 25, что является приемлемым результатом.

После разработки программы было проведено функциональное тестирование, которое помогает проверить работоспособность программы с точки зрения пользователя. Было проверено клиентское приложение, так как для проверки пользовательского интерфейса модульные тесты не подходят.

В приложении есть пять окон: окно списка учётных записей, окно описания учётной записи, окно добавления/редактирования учётной записи, окно настроек, окно регистрации.

В окне списка учётных данных показаны кнопка добавления элемента списка и сами элементы списка. При первой загрузке окно должно подгружать элементы из базы данных. При нажатии на элемент списка должно открываться окно описания элемента. При нажатии кнопку «добавить» открывается окно добавления элемента. При свайпе списка вверх до упора должны подгружаться новые элементы, если они существуют.

В окне описания элемента представлена информация об учётных данных: название, идентификатор (url), логин, пароль. Кроме того, есть кнопка «Удалить» и кнопка «Редактировать». При загрузке окна вся информация должна загружаться и отображаться корректно. Кнопка «Удалить» должна удалять учётную запись из базы данных, выходить в окно списка учётных

данных и обновлять список. Кнопка «Редактировать» должно открывать окно редактирования учётной записи с заполненными полями.

В окне добавления/редактирования учётной записи отображаются следующие поля: название, идентификатор (url), логин, пароль, а также кнопка «Сохранить». Ограничений на содержимое полей нет. Но есть ограничение на сохранение данных. Так при сохранении учётной записи проводится проверка на наличие дубликата в базе данных. При возникновении такой ситуации должно выводиться сообщение об ошибке. При нажатии на кнопку «Сохранить» нового элемента, окно закрывается, открывается окно списка учётных записей с созданной записью. При сохранении существующего элемента, должно открываться предыдущее окно с обновлёнными данными.

В окне настроек есть две кнопки: «Регистрация» и «Синхронизация». При нажатии на кнопку «Синхронизация» происходит синхронизация с сервером и выводиться сообщение об успешности операции, либо сообщение об ошибке. При нажатии на кнопку «Регистрация» открывается окно регистрации.

В окне регистрации есть камера, которая сканирует qr-код. Если qr-код валиден, выводиться сообщение об успешном сканировании. Иначе сообщение об ошибке. При окончании регистрации окно закрывается с сообщением о результате операции.

Все результаты тестирования представлены в таблице 5.

Таблица 5 – Функциональное тестирование приложения

№ теста	Название теста	Ожидаемый результат	Полученный результат	Примечание
1	Загрузка окна списка учётных записей	Показаны имеющиеся учётные записи.	Список пуст.	Элементы загружались до авторизации в хранилище.
2	Добавление новой учётной записи	Открыто окно добавления учётной записи с пустыми полями.	Открыто окно добавления учётной записи с пустыми полями.	
3	Нажатие на элемент списка	Открыто окно описания учётной записи.	Открыто окно описания учётной записи.	

Продолжение таблицы 5

№ теста	Название теста	Ожидаемый результат	Полученный результат	Примечание
4	Свайп вверх	Запущена загрузка новых учётных записей.	Запущена загрузка новых учётных записей.	
5	Удаление учётной записи	Учётная запись удалена из базы. Открыто обновленное окно списка учётных записей.	Учётная запись удалена из базы. Открыто обновленное окно списка учётных записей.	
6	Редактирование учётной записи	Открыто окно редактирования учётной записи с заполненными полями.	Открыто окно редактирования учётной записи с пустыми полями.	Ошибка при переходе с MAUI на Xamarin.
7	Сохранение новой учётной записи	Открыто окно списка учётных записей с новой учётной записью.	Открывается окно списка учётных записей с новой учётной записью.	
8	Сохранение дубликата учётной записи	Показано сообщение, что учётная запись уже существует.	Выбрашено исключение ElementExists Exception	Приложение не падает. Сообщения не видно.
9	Сохранение изменений учётной записи	Открыто предыдущее окно с обновленными полями.	Открыто предыдущее окно с обновленными полями.	
10	Синхронизация с интернетом и регистрацией	Запущен процесс синхронизации с сервером. Выводится сообщение «Success».	Запущен процесс синхронизации с сервером. Выводится сообщение «Success».	
11	Синхронизация без интернета	Показано сообщение об ошибке.	Показано сообщение об ошибке.	
12	Сканирование qr-кода с подключением к локальной сети и подтверждением	Показано сообщение «Success». Открыто предыдущее окно.	Показано сообщение «Success». Открыто предыдущее окно.	
13	Сканирование qr-кода без подключения к локальной сети	Показано сообщение «Fail».	Показано сообщение «Fail».	
14	Сканирование qr-кода без подключения с отклонением	Показано сообщение «Fail».	Показано сообщение «Fail».	

Было обнаружено три ошибки и выявлены причины их возникновения. Все критические ошибки были исправлены.

Для всех остальных проектов разработаны проекты модульных тестов. Они необходимы для проверки работоспособности и корректности выполнения кода. Модульные тесты – это единственный способ проверки работоспособности общих библиотек без разработки клиентского приложения с интерфейсом. Все модульные тесты приведены в таблице 6.

Таблица 6 – Модульное тестирование приложения

№ теста	Название теста
1	тестирование шифрования и дешифрования алгоритмом AES
2	тестирование равенства двух учётных записей
3	тестирование равенства двух учётных сущностей
4	тестирование корректности аутентификации при синхронизации на сервере
5	тестирование корректности полученных данных при синхронизации на сервере
6	тестирование сохранения данных при синхронизации на сервере
7	тестирование аутентификации с ложным устройством при синхронизации на сервере
8	тестирование аутентификации с ложным токеном при синхронизации на сервере
9	тестирование корректности ответа при регистрации устройства на сервере
10	тестирование отмены подтверждения регистрации устройства на сервере
11	тестирование регистрации нескольких устройств в одной сессии
12	тестирование корректной регистрации на сервере
13	тестирование регистрации отказа регистрации на сервере
14	тестирование процесса регистрации с некорректным токеном от устройства
15	тестирование регистрации уже существующего устройства
16	тестирование удаления устройства на сервере
17	тестирование получения списка устройства на сервере
18	тестирование переименования устройства на сервере

Продолжение таблицы 6

№ теста	Название теста
19	тестирование обновления учётной записи в локальной базе данных
20	тестирование сохранения новой учётной записи в локальной базе данных
21	тестирование проверки сохранённого пароля в локальной базе данных
22	тестирование авторизации в локальной базе данных
23	тестирование удаления учётной записи в локальной базе данных
24	тестирование слияния изменений в локальной базе данных

Тесты успешно пройдены, что видно из рисунка 37.

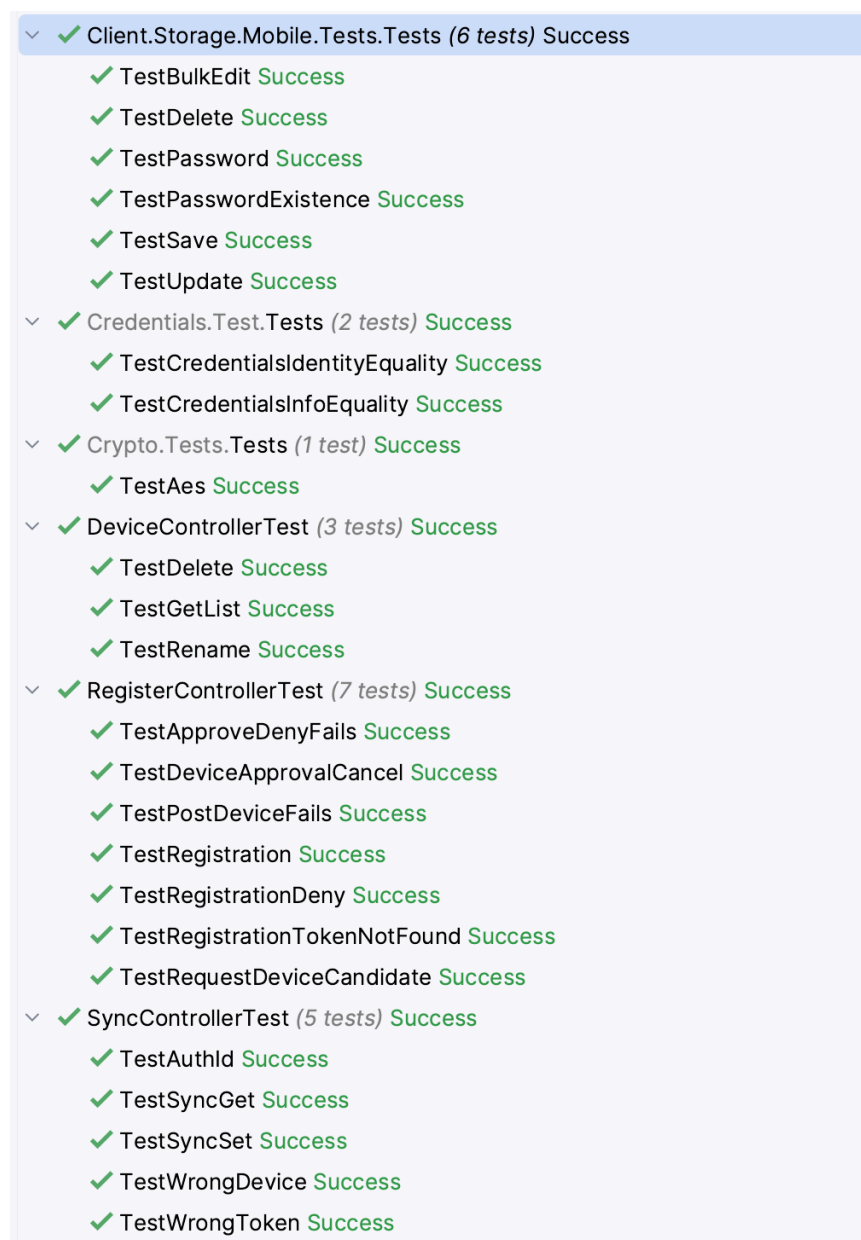


Рисунок 37 – Результат модульного тестирования

Для симуляции коммуникации сервера, клиента и программы администрирования были разработаны интеграционные тесты. Они нужны, чтобы проверить компоненты нескольких приложений, минуя разработку и использования пользовательского интерфейса. В ходе исполнения интеграционных тестов запускается имитационный сервер и выполняется код, отсылающий ему запросы, которые потом будут использоваться в реальных приложениях. Были реализованы следующие тесты:

- тестирование запроса на удаление устройства;
- тестирование запроса на получение списка устройств;
- тестирование запроса на переименование устройства;
- тестирование процесса регистрации;
- тестирование получения данных при синхронизации;
- тестирование отправки и слияния данных при синхронизации.

Тесты успешно пройдены, что видно на рисунке 38.

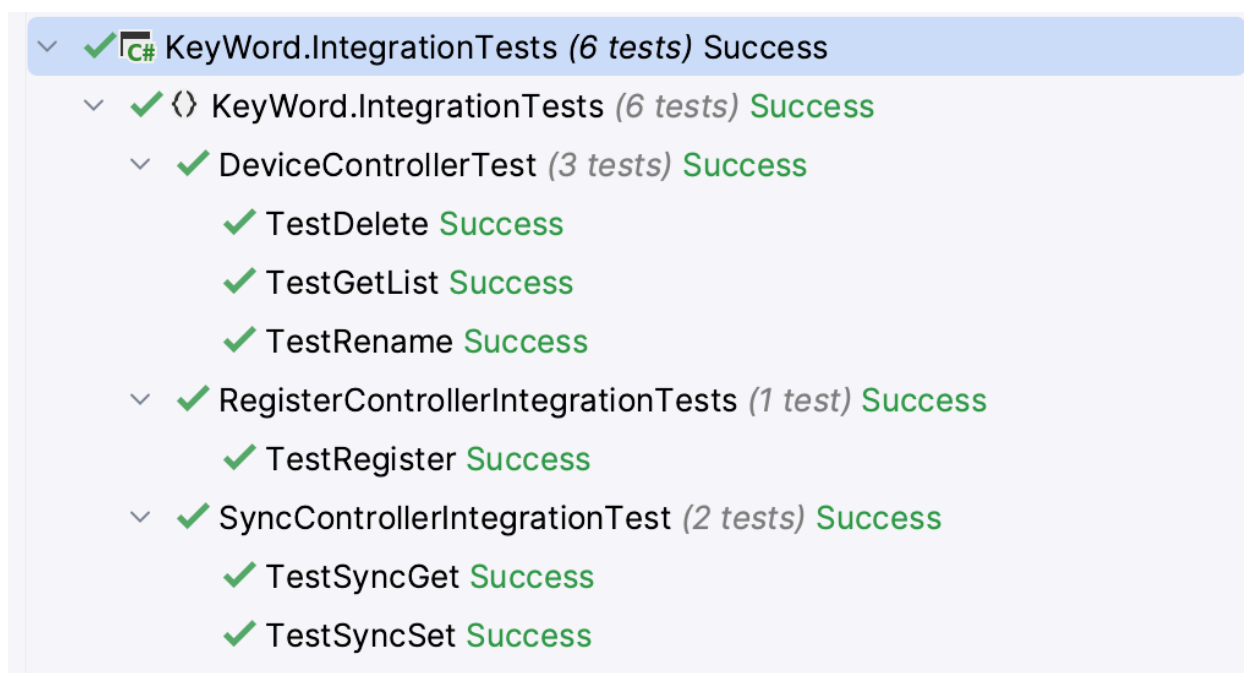


Рисунок 38 – Результат интеграционного тестирования

Таким образом, была осуществлена проверка правильности работы разработанного программной системы при помощи функционального,

модульного и интеграционного тестирования. Все критические ошибки, выявленные в ходе тестирования, были исправлены.

3.3 Аprobация и практическое применение модели и алгоритмов синхронизации

Разработанная система была введена в эксплуатацию на рабочих местах компании ООО «Бит.Геймс Инжиниринг» и используется для хранения и синхронизации учётных данных пользователей полностью внутри компании и на устройствах, принадлежащих ей (приложение А).

До внедрения были использованы аналоги системы, которые отправляют учётные данные в облако: LastPass, Google Password Manager и Apple Keychain. Чтобы получить эти данные дистанционно, необходимо атаковать один из звеньев системы: персональное устройство, канал передачи данных и облако. Атака на последние два звена может быть произведена вне компании и даже государства.

При атаке на канал полученные данные зашифрованы ассиметричным алгоритмом шифрования. Их можно получить в чистом виде, только если известен приватный ключ. Если это так, им можно расшифровать все последующие перехваченные данные, пока пара ключей не будут изменены.

При атаке на облако, можно получить данные в чистом виде, если не используется сквозное шифрование. Риск хранения учётных данных в облаке заключается в том, что оно доступно из открытой сети, там хранится большое количество данных, что привлекает многих злоумышленников. Некоторые из систем-аналогов, которые были использованы в компании ранее, уже страдали от утечек [10].

Более того, у компании нет полного контроля за данными, которые отправляются и хранятся в облаке, но у владельцев облака есть.

Сами пользователи могут случайно выдать свой мастер-паролей, при помощи которого осуществляется доступ к облаку и, тем самым, слить свои пароли.

После внедрения разработанной системы все компоненты системы теперь находятся в пределах и под полным контролем компании и государства. Хранилище учётных данных не доступно из внешней сети. Все данные передаются исключительно по внутренней сети компании, к которой необходимо теперь подключиться злоумышленнику, чтобы перехватить их.

Передаваемые данные шифруются ключом, который меняется при каждой новой синхронизации. Таким образом, получив один ключ шифрования, нельзя расшифровать другие данные.

Пользователи больше не знают свой пароль для доступа к серверу, а аутентификация осуществляется по заранее зарегистрированным устройствам на сервере компании. Даже если злоумышленник получит ключ безопасности, он не сможет получить доступ к данным на сервере.

Для использования разработанного приложения, необходимо его установить. Для ОС «iOS» это делается через «AppStore». Для «Android» можно установить через «Play Market», либо вручную через файл «apk».

Изначально хранилище пустое, и туда можно добавить новую учётную запись. Чтобы это сделать, необходимо нажать на кнопку «Add» (смотри рисунок 33). Откроется новое окно, где требуется ввести данные (смотри рисунок 35). В поле «Name» вводится название учётной записи. Оно может быть любое, но рекомендуется, чтобы в нём присутствовало название ресурса. В поле «URL» вводится адрес интернет-ресурса. Можно вводить не весь адрес страницы, а только доменное имя. В поле «Login» необходимо ввести имя пользователя. В большинстве случаев это почтовый адрес, но есть исключения. В поле «Password» требуется ввести пароль. Он должен соответствовать требованиям, представленным на сайте. Рекомендуется, чтобы длина пароля была не менее 8 символов, содержала буквы как в верхнем, так и в нижнем регистре, цифры и специальные символы. Желательно, чтобы введённый пароль нигде не повторялся. Для удобства, можно составить пароль из нескольких несвязанных фраз, каждая из которых начинается с заглавной буквы, разделена дефисом и оканчивается цифрой или числом. Таким образом, пароль получится

достаточно длинным, будет иметь цифры и специальные символы, а также его можно будет легко ввести с клавиатуры. Запоминать введённый пароль не нужно. После того, как все поля заполнены, учётную запись можно сохранить. Для этого необходимо нажать кнопку «Save». Текущее окно закроется и появится новый элемент списка. При нажатии на него откроется окно описания учётной записи (смотри рисунок 34). Нажатие на поле «Login» и «Password» копируется введённое значение. Нажатие на поле «URL» открывает введённый интернет-ресурс. Кнопка «Remove» удаляет учётную запись. Для редактирования учётной записи, необходимо нажать кнопку «Edit». Откроется окно редактирования, схожее с ранее рассмотренным окном создания.

Чтобы синхронизировать данные между другими устройствами, требуется провести дополнительную настройку. Во-первых, ключ шифрования должен быть передан на другое устройство. Для этого на первом устройстве в разделе настроек, необходимо нажать кнопку «Export Auth Key». Откроется окно с QR-кодом, которое должно быть отсканировано на втором устройстве. Это делается нажатием кнопки «Import Auth Key» в разделе настроек. При передаче ключа шифрования, все данные на устройстве-получателе стираются. После этой процедуры оба устройства готовы к синхронизации.

Чтобы настроить синхронизацию, требуется развернуть сервер синхронизации. Для этого на персональном компьютере устанавливается приложение-сервер и приложение для администрирования. В окне этого приложения на вкладке «Devices» и нажать кнопку «Add» (смотри рисунок 27). Будет показан QR-код, который должен быть отсканирован на обоих устройствах (смотри рисунок 28). Для этого на устройстве в разделе настроек нажимается кнопка «Register Device», а затем на персональном компьютере подтверждается регистрацию. После этого в окне устройств появятся добавленные устройства, которые можно удалить или переименовать. На добавленных устройствах можно синхронизировать данные. Для этого в разделе учётных записей необходимо пролистать вверх до упора или в разделе настроек нажать кнопку «Synchronize». Синхронизация работает только, когда сервер и устройства находятся в одной

сети. Это может быть локальная сеть или сеть VPN. Во втором случае синхронизация возможна по сети Интернет, но это не рекомендуется делать.

Таким образом были существенно снижены риски утечки учётных данных сотрудников и риски несанкционированного доступа к важным приложениям компании.

Выводы

Разработанные модель и алгоритмы были реализованы с помощью выбранного технологического стека. Реализованный проект прошёл проверку ручным, модульным и интеграционным тестами. Выявленные критические ошибки исправлены.

Заключение

В ходе магистерской диссертации были решены следующие задачи:

- 1) проанализированы методы обеспечения безопасности хранения и передачи учётных данных пользователей;
- 2) разработаны графические модели хранения учётных данных и механизм их синхронизации;
- 3) разработаны алгоритмы синхронизации учётных данных пользователей интернет-ресурсов и приложений;
- 4) на основании моделей и алгоритмов разработаны программные средства хранения и передачи учётных данных.

В данной работе был проведен анализ существующих моделей и алгоритмов хранения и передачи учётных данных. Были выявлены векторы атаки и риски утечки.

В ходе научно-исследовательской работы были разработаны новые модели и алгоритмы хранения и синхронизации учётных данных, а также алгоритмы регистрации устройств. На новых моделях возможно меньшее количество атак и рисков, чем на существующих.

На основе разработанных моделей и алгоритмов создана, протестирована и отлажена программная реализация хранения и передачи учётных данных пользователей. Результат магистерской работы внедрён в эксплуатацию.

Список используемых источников

1. Джереми Кейт. The password anti-pattern // adactio [Электронный ресурс] URL: <https://adactio.com/journal/1357/> (дата обращения: 20.04.2021)
2. Слабые пароли могут вам слишком дорого обойтись // Avast [Электронный ресурс] <https://blog.avast.com/ru/slabye-paroli-mogut-vam-slishkom-dorogo-obojtis> (дата обращения: 20.04.2021).
3. Лим, В.Б. Создание надежных паролей / В.Б. Лим. // Проблемы науки. 2021. № 3 (62). С. 23-24.
4. Тереньтьев, А.М. Выбор адекватных средств информационной защиты персонального компьютера в России / А.М. Тереньтьев. // Национальные интересы: приоритеты и безопасность. 2012. №2 (15). С. 37-42.
5. Абидарова, А.А. Анализ надежности паролей для обеспечения информационной безопасности / А.А. Абидарова // Известия Тульского государственного университета. Технические науки. 2021. №8. С. 66-68.
6. Абдуллаева, М.А. Проектирование эффективной информационной программы обеспечения защиты информации / М.А. Абдуллаева, Р. Мирзоев. // Перспективы развития информационных технологий. 2014. №21. С. 132-136.
7. Лысенко, А. В. Анализ атак man in the middle / А. В. Лысенко, И. С. Кожевникова, Е. В. Ананьин [и др.]. // Молодой ученый. 2016. № 30 (134). С. 33-36.
8. How it works – Dashlane // [Электронный ресурс]. URL: <https://www.dashlane.com/features>. (дата обращения: 20.04.2021).
9. How it works – LastPass // [Электронный ресурс]. URL: <https://www.lastpass.com/how-lastpass-works> (дата обращения: 20.04.2021).
10. LastPass Hacked – Identified Early & Resolved // Blog LastPass [Электронный ресурс]. URL: <http://blog.lastpass.com/2015/06/lastpass-security-notice/> (дата обращения: 20.04.2021).
11. KeePass features [Электронный ресурс] // URL: <https://keepass.info/features.html> (дата обращения: 24.04.2021).

12. MacOS X Security [Электронный ресурс] // URL: https://web.archive.org/web/20110410065128/https://images.apple.com/macosx/security/docs/MacOSX_Security_TB.pdf (дата обращения: 24.04.2021).
13. A Comparison of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish for Guessing Attacks Prevention [Электронный ресурс] // Journal of Computer Science Applications and Information Technology. URL: <https://symbiosisonlinepublishing.com/computer-science-technology/computerscience-information-technology32.php> (дата обращения: 10.01.2022).
14. Intel® AES New Instructions Architecture [Электронный ресурс] // Intel® Advanced Encryption Standard (AES) New Instructions Set. URL: <https://www.intel.com/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf> (дата обращения: 10.01.2022).
15. “S-Box” Implementation of AES is NOT side channel resistant [Электронный ресурс] // URL: <https://eprint.iacr.org/2018/1002.pdf> (дата обращения: 10.01.2022).
16. Deep-Learning Side-Channel Attacks on AES [Электронный ресурс] // URL: <https://www.diva-portal.org/smash/get/diva2:1322924/FULLTEXT01.pdf> (дата обращения: 10.01.2022).
17. A Meet in the Middle Attack on Reduced Round Kuznyechik [Электронный ресурс] // URL: <https://eprint.iacr.org/2015/096.pdf> (дата обращения: 10.01.2022).
18. Don't use bcrypt [Электронный ресурс] // URL: <http://www.unlimitednovelty.com/2012/03/dont-use-bcrypt.html> (дата обращения: 10.01.2022).
19. Stronger key derivation via sequential memory-hard functions [Электронный ресурс] // URL: <http://www.tarsnap.com/scrypt/scrypt.pdf> (дата обращения: 10.01.2022).

20. Implementing AutoFill Credential Provider Extensions // Developer Apple [Электронный ресурс] URL: <https://developer.apple.com/videos/play/wwdc2018/721/> (дата обращения: 20.04.2021).
21. Введение в ORM // Блог веб-разработчика [Электронный ресурс]. URL: <http://internetka.in.ua/orm-intro/> (дата обращения: 20.04.2021).
22. Самсонкин, А.С. Обзор и сравнительный анализ кроссплатформенных средств разработки / Самсонкин А.С., Такташкин Д.В. // информационные технологии в науке и образовании. Проблемы и перспективы. Сборник статей по материалам VIII Всероссийской межвузовской научно-практической конференции. Пенза. 2021. С. 178-180.
23. Самсокин, А.С. KeyWord / А.С. Самсонкин // GitHub [Электронный ресурс]. URL: <https://github.com/wrenge/KeyWord> (дата обращения: 20.04.2022).

Приложение А

АКТ О ВРЕДНЕНИИ РЕЗУЛЬТАТОВ МАГИСТЕРСКОЙ ДИССЕРТАЦИИ
(рекомендуемое)



УТВЕРЖДАЮ
Технический директор
ООО «Бит.Геймс Инжиниринг»


П. А. Щеваев
«31» мая 2023

АКТ

Внедрения результатов магистерской диссертации
«Синхронизация учётных данных пользователей интернет-ресурсов
и приложений»
Самсонкина Андрея Сергеевича

Результаты магистерской диссертации Самсонкина Андрея Сергеевича «Синхронизация учётных данных пользователей интернет-ресурсов и приложений» внедрены в производственную эксплуатацию

В данной работе была разработана графическая модель хранения и передачи учётных данных, отличающийся от существующих отсутствием в системе устройств третьих лиц, позволяющая хранить и передавать данные более децентрализованно. Данная модель помогла уменьшить риски утечки паролей и конфиденциальных данных и риски несанкционированного доступа к элементам управления компании.

Разработан алгоритм синхронизации данных, отличающийся от существующих беспарольной системой защиты, и не позволяющий третьим лицам получить несанкционированный доступ к паролю пользователя. В системе учтены возможные недостатки аналогов, рассмотрен представленный алгоритм и внедрён на рабочих местах компании.

Приложение Б
ПОДРОБНОЕ ОПИСАНИЕ ПРЕЦЕДЕНТОВ
(рекомендуемое)

Таблица Б.1 – Прецедент «развернуть сервер»

Прецедент: Развернуть сервер
ID: 1
Краткое описание: Разворачивает сервер на локальной машине
Главные актёры: Пользователь
Второстепенные актеры: Нет.
Предусловия: Нет.
Основной поток: 1. Пользователь устанавливает сервер-демон 2. Пользователь устанавливает приложения для управления сервером
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица Б.2 – Прецедент «зарегистрировать новое устройство»

Прецедент: Зарегистрировать новое устройство
ID: 2
Краткое описание: Добавляет устройство в список зарегистрированных на сервере
Главные актёры: Пользователь
Второстепенные актеры: Нет.
Предусловия: 3. Приложение запущено 4. Сервер запущен 5. У приложения есть доступ к серверу по локальной сети 6. Устройство не зарегистрировано
Основной поток: 1. Пользователь на сервере добавляет устройство 2. На экране сервера появляется QR-код 3. Пользователь сканирует QR-код 4. Пользователь подтверждает добавление устройство 5. Устройство получает от сервера необходимые ключи
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица Б.3 – Прецедент «синхронизировать учётные данные»

Прецедент: Синхронизировать учётные данные
ID: 3
Краткое описание: Выполняет синхронизацию учётных данных с сервером
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Программа запущена 2. Вход авторизован 3. Устройство зарегистрировано на сервере 4. Есть доступ к серверу
Основной поток: 1. Пользователь нажимает кнопку «синхронизировать» 2. Находятся несинхронизированные записи 3. Отправляется запрос на сервер 4. Сервер отправляет свои несинхронизированные записи
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица Б.4 – Прецедент «удалить учётные данные»

Прецедент: Удалить учётные данные
ID: 4
Краткое описание: Удаляет одну запись учётных данных
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Программа запущена 2. Вход авторизован
Основной поток: 1. Пользователь нажимает кнопку «удалить» 2. Происходит удаление учётной записи 3. Запускается синхронизация
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица Б.5 – Прецедент «обновить учётные данные»

Прецедент: Обновить учётные данные
ID: 5
Краткое описание: Вносит изменение в уже имеющуюся учётную запись
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Программа запущена 2. Вход авторизован
Основной поток: 1. Пользователь нажимает кнопку «сохранить» 2. Учётные данные шифруются 3. Происходит сохранение изменений учётной записи 4. Запускается синхронизация
Постусловия: Нет.
Альтернативные потоки: Нет.

Таблица Б.6 – Прецедент «сохранить учётные данные»

Прецедент: Сохранить учётные данные
ID: 6
Краткое описание: Сохраняет учётные данные в хранилище
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Программа запущена 2. Вход авторизован
Основной поток: 1. Пользователь нажимает кнопку сохранить 2. Проверяется мастер-пароль 3. Учётные данные шифруются 4. Учётные данные сохраняются в хранилище 5. Запускается синхронизация
Постусловия: Нет.
Альтернативные потоки: Нет.

Приложение В

АЛГОРИТМ РЕГИСТРАЦИИ УСТРОЙСТВА (рекомендуемое)

Листинг В.1 – Алгоритм регистрации нового устройства на сервере

```

[ApiController]
[Route("[controller]")]
public class RegisterController : ControllerBase
{
    private const int TokenByteLength = 32;
    // private const int TokenTimeout = 30;
    private const int TokenTimeout = 600;
    private readonly RegisterService _registerService;
    private readonly ILogger<RegisterController> _logger;
    private readonly IStorage _storage;
    private RegisterSession? CurrentSession
    {
        get => _registerService.CurrentSession;
        set => _registerService.CurrentSession = value;
    }

    public RegisterController(IStorage storage, RegisterService
registerService, ILogger<RegisterController> logger)
    {
        _storage = storage;
        _logger = logger;
        _registerService = registerService;
    }

    // Only from admin client
    [Host("localhost")]
    [HttpPost(nameof(StartNewRegistration))]
    public ActionResult StartNewRegistration()
    {
        if (CurrentSession is {IsClosed: false, IsExpired: false })
        {
            _logger.LogWarning( "Closing session early: {Token}",
CurrentSession.Token);
            CurrentSession.Close();
        }

        var newToken =
Convert.ToBase64String(RandomNumberGenerator.GetBytes(TokenByteLen
gth));
        var port = NetworkConstants.RequestPort; // TODO
генерировать рандомный порт
        CurrentSession = new RegisterSession(newToken,
DateTime.Now, TimeSpan.FromSeconds(TokenTimeout), port);
        _logger.LogInformation( "Starting session {Token}",
CurrentSession.Token);
        Task.Run(async () => await
CurrentSession.ListenDiscoveryAsync());
        Task.Run(async () => await
CloseSessionOnExpire(CurrentSession));
        return Ok();
    }
}

```

```

// Only from admin client
[Host("localhost")]
[HttpPost(nameof(CloseSession))]
public ActionResult CloseSession()
{
    if (CurrentSession is {IsClosed: false, IsExpired: false })
    {
        CurrentSession.Close();
    }

    return Ok();
}

private async Task CloseSessionOnExpire(RegisterSession
session)
{
    await Task.Delay(session.GetTimeLeft());
    _logger.LogInformation("Closing session {Token}",
session.Token);
    session.Close();
}

// Only from admin client
[Host("localhost")]
[HttpGet(nameof(RequestNewToken))]
public ActionResult<RegisterInfo> RequestNewToken()
{
    if (CurrentSession == null || CurrentSession.IsExpired ||
CurrentSession.IsClosed)
    {
        return NotFound();
    }
    return new RegisterInfo(CurrentSession.Token,
CurrentSession.GetExpireDate(), CurrentSession.Port);
}

// Only from admin client
[Host("localhost")]
[HttpGet(nameof(RequestDeviceCandidate))]
public async Task<ActionResult<DeviceCandidate?>>
RequestDeviceCandidate()
{
    if (CurrentSession == null || CurrentSession.IsClosed)
    {
        const string errorMsg = "Tried to await device without
starting a session";
        _logger.LogError(errorMsg);
        return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
    }

    if (CurrentSession.IsExpired)

```

```

        {
            const string errorMsg = "Tried to await device with
expired session";
            _logger.LogError(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }

        try
        {
            var device = await CurrentSession.DeviceCandidate.Task;
            // TODO: if connection interrupted then close session
            return device;
        }
        catch (TaskCanceledException)
        {
            return
StatusCode(StatusCodes.Status503ServiceUnavailable);
        }
    }

    // Only from admin client
    [Host("localhost")]
    [HttpPost(nameof(ApprovePendingDevice))]
    public ActionResult ApprovePendingDevice()
    {
        if (CurrentSession == null || CurrentSession.IsClosed)
        {
            const string errorMsg = "Tried to approve device without
session";
            _logger.LogInformation(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }

        if (!CurrentSession.DeviceCandidate.Task.IsCompleted)
        {
            const string errorMsg = "Tried to approve device without
device itself";
            _logger.LogInformation(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }

        if (CurrentSession.IsExpired)
        {
            const string errorMsg = "Tried to approve device with
an expired session";
            _logger.LogInformation(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }
    }

```

```

        CurrentSession.IsDeviceApproved = true;
        CurrentSession.Close();
        var deviceCandidate =
CurrentSession.DeviceCandidate.Task.Result;
        _storage.AddDevice(new Device()
        {
            Id = deviceCandidate.Id,
            Name = deviceCandidate.Name,
            RegisterDate = DateTime.Now,
            Token = deviceCandidate.Token
        });
        return Ok();
    }

    // Only from admin client
    [Host("localhost")]
    [HttpPost(nameof(DenyPendingDevice))]
    public ActionResult DenyPendingDevice()
    {
        if (CurrentSession == null || CurrentSession.IsClosed)
        {
            const string errorMsg = "Tried to deny device without
session";
            _logger.LogInformation(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }

        if (CurrentSession.IsExpired)
        {
            const string errorMsg = "Tried to deny device with an
expired session";
            _logger.LogInformation(errorMsg);
            return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
        }

        CurrentSession.IsDeviceApproved = false;
        CurrentSession.Close();
        return Ok();
    }

    // Only from app client
    [HttpPost(nameof(PostDeviceInfo))]
    public ActionResult PostDeviceInfo([FromBody] DeviceCandidate
device)
    {
        if (CurrentSession == null || CurrentSession.IsClosed)
        {
            const string errorMsg = "Tried to post device info
without session";
            _logger.LogInformation(errorMsg);

```

```

        return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
    }

    if (device.Token != CurrentSession.Token)
    {
        const string errorMsg = "Tried to post device info with
an invalid token";
        _logger.LogInformation(errorMsg);
        return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
    }

    if (CurrentSession.IsExpired)
    {
        const string errorMsg = "Tried to post device info with
an expired session";
        _logger.LogInformation(errorMsg);
        return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
    }

    if (CurrentSession.IsOccupied)
    {
        const string errorMsg = "Tried to post device info with
an occupied token";
        _logger.LogInformation(errorMsg);
        return Problem(errorMsg, statusCode:
StatusCodes.Status400BadRequest);
    }

    CurrentSession.IsOccupied = true;
    CurrentSession.DeviceCandidate.SetResult(device);
    return Ok();
}

// Only from app client
[HttpGet(nameof(GetDeviceApproval) +("/{idUrlEncoded}"))]
public async Task<ActionResult> GetDeviceApproval(string
idUrlEncoded)
{
    var deviceId = HttpUtility.UrlDecode(idUrlEncoded);
    var existingDevice = _storage.FindDeviceById(deviceId);
    if (existingDevice != null)
    {
        return Ok();
    }

    if (CurrentSession == null
        || CurrentSession.IsClosed
        || CurrentSession.IsExpired
        || !CurrentSession.DeviceCandidate.Task.IsCompleted

```

```

        || CurrentSession.DeviceCandidate.Task.Result.Id !=
deviceId
    )
    {
        return Forbid();
    }

    try
    {
        await CurrentSession.GetApprovalHandle();
        return Ok();
    }
    catch (TaskCanceledException)
    {
        return Forbid();
    }
}
}

```

Листинг В.2 – Алгоритм регистрации нового устройства на устройстве

```

public class RegisterService : INetworkService
{
    private readonly HttpClient _client;

    public RegisterService(HttpClient client)
    {
        _client = client;
    }

    public RegisterService(Uri baseAddress)
    {
        _client = new HttpClient();
        _client.BaseAddress = baseAddress;
    }

    public async Task<bool> TryRegister(string id, string name,
string token)
    {
        var deviceInfo = new DeviceCandidate()
        {
            Id = id,
            Name = name,
            Token = token
        };
        var json = JsonSerializer.Serialize(deviceInfo);
        var content = new StringContent(json, Encoding.UTF8,
MediaTypeNames.Application.Json);
        var postResponse = await
_client.PostAsync("/Register/PostDeviceInfo", content);
        if (!postResponse.IsSuccessStatusCode)
            return false; // TODO детализировать ошибку
    }
}

```

```
var idUrlEncoded =  
HttpUtility.UrlEncode(deviceInfo.Id);  
var approvalResponse = await  
_client.GetAsync($"Register/GetDeviceApproval/{idUrlEncoded}");  
return approvalResponse.IsSuccessStatusCode;  
}  
}
```

Приложение Г

АЛГОРИТМ синхронизации учётных данных (рекомендуемое)

Листинг Г. 1 – Алгоритм синхронизации данных на сервере

```

public ActionResult<SyncResponse?> RequestSync([FromBody]
SyncRequest request)
{
    var device = Storage.FindDeviceById(request.DeviceId);
    if (device == null)
        return Unauthorized("Device is not registered");

    var keyValid = request.AuthKey ==
SyncUtilities.GetDeviceAuthKey(device.Id, device.Token).ToBase64();

    if(!keyValid)
        return Unauthorized("Auth key is not valid");

    var authId = request.AuthId;
    if (string.IsNullOrEmpty(authId))
        return Unauthorized("AuthId is empty");

    var syncData = new SyncData
    {
        AddedCredentials =
Storage.GetAddedCredentials(request.LastSyncTime,
authId).ToArray(),
        ModifiedCredentials =
Storage.GetModifiedCredentials(request.LastSyncTime,
authId).ToArray(),
        DeletedCredentialsIds =
Storage.GetDeletedCredentials(request.LastSyncTime,
authId).ToArray()
    };

    Storage.AddCredentials(request.SyncData.AddedCredentials,
authId);

    Storage.UpdateCredentials(request.SyncData.ModifiedCredentials,
authId);

    Storage.DeleteCredentials(request.SyncData.DeletedCredentialsIds,
authId);

    var result = new SyncResponse();
    result.SyncData = syncData;

    return result;
}

```

Листинг Г. 2 – Алгоритм синхронизации данных на устройстве

```

public async Task<SyncResponse?> TrySync(string deviceId,
string deviceToken,
string authId,
DateTime lastSyncTime,

```

```

        IEnumerable<ClassicCredentialsInfo> added,
        IEnumerable<ClassicCredentialsInfo> modified,
        IEnumerable<int> deleted)
    {
        var authKey = SyncUtilities.GetDeviceAuthKey(deviceId,
deviceToken).ToBase64();
        var syncRequest = new SyncRequest
        {
            DeviceId = deviceId,
            AuthKey = authKey,
            AuthId = authId,
            LastSyncTime = lastSyncTime,
            SyncData = new SyncData
            {
                AddedCredentials = added.ToArray(),
                ModifiedCredentials = modified.ToArray(),
                DeletedCredentialsIds = deleted.ToArray()
            }
        };

        var json = JsonSerializer.Serialize(syncRequest);
        var content = new StringContent(json, Encoding.UTF8,
"application/json");
        var postResponse = await
_client.PostAsync("Sync/RequestSync", content);
        if (!postResponse.IsSuccessStatusCode)
            return null;

        var responseContent = await
postResponse.Content.ReadAsStringAsync();
        var syncResponse =
        JsonSerializer.Deserialize<SyncResponse>(responseContent,
        new JsonSerializerOptions(JsonSerializerDefaults.Web));
        return syncResponse;
    }

```