Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерного проектирования
Кафедра Проектирования информационно-компьютерных систем

К защите допустить:

Заведующий кафедрой ПИКС

_____ И. Н. Цырельчук

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

СЕТЕВОЙ МЕНЕДЖЕР ПАРОЛЕЙ С ШИФРОВАНИЕМ/ДЕШИФРОВАНИЕМ ДАННЫХ

БГУИР ДП 1-39 03 01 011 ПЗ

Студент	А. П. Деревнюк
Руководитель	В. М. Логин
Консультанты:	
от кафедры	В. М. Логин
по экономической части	А. В. Слюсарь
Нормоконтролёр	В. В. Хорошко

Рецензент

Минск 2016

СОДЕРЖАНИЕ

Введение 7
1 Обзор литературы 8
1.1 Обзор существующих программ 8
1.2 Оценка стойкости шифра в системах с открытым ключом . . . 10
2 Системное проектирование 12
2.1 Анализ предметной области 12
2.2 Идентификация объектов предметной области 14
2.3 Декомпозиция объектов предметной области 15
3 Функциональное проектирование 18
3.1 Описание конфигурации сборки проекта 18
3.2 Классы Business Layer 22
3.3 Классы Data Layer 30
3.4 Классы Application Server Layer 40
4 Разработка программных модулей 45
4.1 Алгоритм расчета апостериорной вероятности для оценки стойкости шифрования 45
4.2 Классификатор нормализации данных 51
5 Программа и методика тестирования программного продукта . . . 56
5.1 Ручное тестирование 57
5.2 Unit-тестирование 58
6 Руководство пользователя 59
6.1 Авторизация пользователя в системе 59
6.2 Управление группами в хранилище 61
6.3 Управление записями в группе 62
7 Технико-экономическое обоснование эффективности разработки и использования сетевого менеджера паролей с шифрованием/дешифрованием данных 64
7.1 Характеристика программного продукта 64
7.2 Расчет стоимостной оценки затрат 65
7.3 Расчет затрат на разработку и отпускной цены программного продукта 65
7.4 Расчет стоимостной оценки результата 70
7.5 Расчет показателей экономической эффективности проекта . . 71
Заключение 75

Список использованных источников 76
Приложение А (обязательное) Исходный текст класса Archive 78
Приложение Б (обязательное) Исходный текст класса ManagedGroup . 83
Приложение В (обязательное) Исходный текст класса EventEmittor . . 89
Ведомость документов 99

ВВЕДЕНИЕ

На момент разработки дипломного проекта, как сообщает американская компания по организации информационной безопасности Hold Security [1], с начала текущего года лишь одной группой хакеров было похищено более двух миллионов различных пользовательских данных.

Несмотря на то, что существует программное обеспечение, которое способно решить эту проблему, пользователи предпочитают хранить свои данные на общедоступных ресурсах, подвергая их риску быть похищенным. Особенностью разрабатываемого приложения является криптосистема, отвечающая современным стандартам [2], и использование собственных алгоритмов оптимизации скорости вычисления шифров.

Использование менеджеров паролей является обязательным условием для эффективного управления и хранения пользовательских данных. Это обусловлено единством интерфейсов и полным сокрытием реальных данных, посредством шифрования.

Вопрос коммерциализации подобного програмного обеспечения находится в стадии становления. Главной задачей его участников является развитие клиентской базы и превращение технологических идей в прибыльный бизнес. При этом необходимо решить множество организационных, технологических и финансовых вопросов.

Все продукты, направленные на управление данными пользователя, различаются по многим параметрам: временным и трудовым затратам, сложностью, требуемой надежности системы и так далее. Эти параметры влияют на стоимость разработки и сложность взаимодействия. Для успешной реализации любого крупного проекта недостаточно только выбрать эффективную технологию и средство разработки.

Главной задачей, при разработки дипломного проекта, ставился вопрос оптимизации существующих алгоритмов шифрования и оценки криптографической стойкости системы; создания кроссплатформной архитектуры, с полной поддержкой всего функционала в различных средах использования, включая любую общедоступную сеть.

Целью дипломного проекта является реализация сетевого менеджера паролей с шифрованием данных. Задачей проекта ставится исправление известных недостатков существующих модулей криптографии, применяемых в сетевых системах со сложным интерфейсом администрирования, создание простого, наглядного интерфейса, а так же примесь новых средств для управления данными.

1 ОБЗОР ЛИТЕРАТУРЫ

Целью литературного обзора является исключение неоправданного дублирования исследований и разработок, глубокое изучение и широкое использование последних достижений науки и техники в отрасли, обеспечение конкурентоспособности и высокого технического уровня объектов разработки. Выбор пути оптимального решения задачи предполагает отбор лучших разработок, которые могут быть использованы в качестве прототипа или в виде модуля в собственной разработке.

Проведению литературно обзора предшествует просмотр научно -технической и популярной литературы, анализ проектов с открытым исходным кодом в популярных системах контроля версий.

Основной задачей литературно обзора является выяснение того, как решаются аналогичные задачи другими разработчиками, а также в каком направлении следует проводить исследования в данной области разработки.

С учётом темы дипломного проекта, организация поиска проводилась по тематике достижений в области криптографических систем и других систем хранения и защиты пользовательских данных.

1.1 Обзор существующих программ

Существует множество менеджеров паролей с шифрованием/дешифрованием данных, но лишь не многие из них представляют веб-интерфейс для удобной работы в сети. Ниже приведены некоторые из задач, которые может выполнять типичный менеджер паролей:

– ручное, автоматическое, полу-автоматическое создание сертификатов доступа к хранилищу;

– оценку параметров условных распределений записей по представленным данным;

– реализацию статистического вывода суждений о безопастности хранения предоставленных данных;

– работу с группами записей и группами хранилищ;

– предоставление доступа к шифрованным данным и безопасное хранение, предоставленной пользователем, информации;

– предоставление доступа к функционалу приложения посредством веб-интерфейса.

Приняв во внимание тему дипломного проекта, наибольший интерес в существуем программном обеспечении будет представлять функциональность предоставление доступа к шифрованным данным и их безопасное хра-

нение. Ниже рассматриваются некоторые из программ реализующие решение описанных выше задач.

1.1.1 LastPass[1)] довольно старый менеджер паролей. Весь его функционал по управлению паролями реализован через веб-приложение и через браузерные плагины. База данных с паролями шифруется с помощью AES-256 и синхронизируется между хранилищем плагина и сервером LastPass. Есть также portable-версии, причем как браузерных плагинов, так и самостоятельного приложения под Windows. Также стоит отметить, что мобильные приложения есть для любой мобильной платформы. Более того, например, для Android есть как отдельное приложение, так и плагин для браузера Dolphin [3]. Версия программы с ограниченной функциональностью свободно доступна на сайте LastPass.

1.1.2 My1login[2)] позволяет распространять записи хранилища по общедоступным API. В основе лежит веб-приложение, с возможностью редактировать записи, а для сохранения их из форм и автоматического ввода существует JavaScript букмарклет. Основной возможностей менеджера паролей является групповая работа с записями хранилища. Подобный функционал будет полезен группам, которые имеют внутри себя определенную базу реквизитов. В данном случае, после плановой смены пароля к определенному сервису отпадет необходимость в сообщении новых данных каждому пользователю. Обладает уникальной моделью двухступенчатой авторизации. В отличии от Lastpass, описанного в подразделе 1.1.1, не предоставляет доступ к удаленному хранилищу, что затруднет его использование на различных платформах.

1.1.3 Bluepass[3)] является менеджером паролей с открытым исходным кодом. Он реализован на Python, что позволило разработчикам создать кроссплатформенное приложение для десктопных клиентов. Обеспечивает синхронизацию по протоколу P2P, тем самым снижая вероятность массовой утечки реквизитов. На текущий момент не имеет веб-интерфейс для работы в сети.

[1)] `https://lastpass.com/`
[2)] `https://www.my1login.com/`
[3)] `https://bluepass.org/`

1.2 Оценка стойкости шифра в системах с открытым ключом

Стойкость систем с открытым ключом основывается на большой вычислительной сложности известных алгоритмов разложения числа на простые сомножители. Выражение для разложения числа согласно основной теореме арифметики в общем случае представлено формулой (1.1):

$$\begin{aligned}\Phi(n) &= \Phi\left(p_1^{q1}\right) \cdot \Phi\left(p_2^{q2}\right) \cdots \Phi\left(p_k^{qk}\right) = \\ &= p_1^{q1} \cdot \left(1 - \frac{1}{p_1}\right) \cdot p_2^{q2} \left(1 - \frac{1}{p_2}\right) \cdots p_k^{qk} \left(1 - \frac{1}{p_k}\right), \end{aligned} \tag{1.1}$$

где Φ — функция Эйлера — мультипликативная арифметическая функция, равная количеству натуральных чисел;

p_k и q — простые числа.

Следует отметить, что p и q выбираются таким образом, чтобы n было больше кода любого символа открытого сообщения. В автоматизированных системах исходное сообщение переводиться в двоичное представление, после чего шифрование выполняется над блоками бит равной длины. При этом длина блока должна быть меньше, чем длина двоичного представления n.

Несмотря на то, что доказательство гипотезы [4, с. 26 – 30], которая лежит в основе функции Эйлера в криптографических системах, о том, что нет такого значения m, которое функция Эйлера принимала бы только один раз, содержит ошибку $\dim(\varphi^{-1}(m)) = 1, n > 10^{37}$, ее продолжают использовать в различных современных сетевых системах, требующих минимальное количество затрат ресурсов.

Для решения ошибки факторизации функции Эйлера во многих системах используется простой алгоритм перебора делителей [5], путем полного перебора всех возможных потенциальных делителей. Суть алгоритма заключается в переборе всех целых чисел i от двух до $\sqrt{n}$ и вычислении остатка от деления $n \mod i$. Если остаток от деления равен нулю, то i является делителем n. В случае, если $i > \sqrt{n}$, то n является простым сомножителем.

Для осуществления поиска всех простых делителей числа n необходимо использовать описанный выше алгоритм рекурсивно, для каждого вновь найденного делителя.

В связи с этим, в рамках системы с асинхронным шифрованием, был разработана реализация алгоритма К2, приведенный в работе [6], псевдокод которого показан в листинге 4.1. Он использует вероятностные сети, как од-

ним из возможных способов представления совместного распределения множества простых чисел. Здесь и далее под назначением простых чисел $X_1, X_2, \ldots, X_n$ понимаются определенные значения, которые принимают случайные величины. Табличное представление совместного распределения растет экспоненциально количеству переменных и состояний, которые эти переменные могут принимать. Благодаря информации о независимости, распределение $P(X_1, X_2, \ldots, X_n)$ может быть факторизовано более просто, чем с использованием функции Эйлера $P(X_1, X_2, \ldots, X_n) = P(X_1)P(X_2|X_1) \cdots P(X_n|X_1, \times \times, X_{n-1})$. При наличии информации о данных, подвергаемым кодировке, совместное распределение случайных величин может быть факторизовано по формуле:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=0}^{n} P(X_i|X_{\pi_i}), \tag{1.2}$$

где π_i — множество индексов переменных-родителей для переменной X_i.

Данное представление совместного распределения все так же имеет экспоненциальный рост количества стохастических параметров от количества переменных и их состояний. Но на практике, обычно они имеют небольшую связанность, далекую от $\dim(\varphi^{-1}(m)) = 1, n > 10^{37}$, что позволяет представлять простые числа в сетевых системах криптографии.

Необходимо отметить, что существует пример успешного коммерческого применения одной из модификаций подобного подхода. К сожалению, работы, в которых данные сети были бы формализованы и математически доказана их корректность, пока не публиковались в открытых источниках. Отличие данной модификации от классических сети заключается в том, что таблицы условных распределений $P(X_k|X_{\pi_k})$, где k — индекс случайной величины, ассоциированные с вершинами графа и имеющими размерность разрядности искомого простого числа $\alpha_k \cdot \prod_{j \in \pi_k} \alpha_j$, заменяются на n таблиц меньшего размера, представляющих условные распределения $P(X_k|X_j)$, где $j \in \pi_k$, $k = 1, \ldots, n$, ассоциированных с дугами графа и имеющими общую размерность $\alpha_k \cdot \sum_{j \in \pi_k} \alpha_j$, где α_j — количество значений, которые может принимать случайная величина X_j.

Разработанная в рамках дипломного проекта реализация сети предназначена для работы именно с таким алгоритмом. Он позволяет в в рамках запроса пользователя определить стойкость системы, на основании переданных данных, решая проблему факторизации, существующую для данных систем.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

В данном разделе будет произведён обзор предметной области задачи, решаемой в рамках дипломного проекта; рассмотрен вопрос об архитектуре разработанного программного обеспечения. Также будут рассмотрены принципы работы структурных блоков, реализованных в программном обеспечении, разработанном в рамках дипломного проекта.

2.1 Анализ предметной области

Как уже упоминалось выше, целью настоящего дипломного проекта является создание сервиса и клиентского приложения к нему, предоставляющего возможность хранить и редактировать различные пользовательские данные, предотвращая их утечку третьим лицам, скрывая их сигнатуру.

Клиентское приложение взаимодействует с сервером, посредством API. Сервис по взаимодействию с базой данных поставляется вместе с драйвером СУБД и инкапсулирует всю логику. Клиент должен предоставлять интерфейс, с помощью которого пользователь сможет получать и сохранять информацию, предоставляемую сервисом.

Важным моментом при разработке приложения является то, что система является много-компонентной. К разным компонентам системы может применяться различные архитектурные подходы. В связи с чем, модели данных, представление информации и протоколы взаимодействия не являются однотипными, поэтому архитектурная надежность программного обеспечения, обеспечивающего взаимодействие и обмен данными, является немаловажным фактором при проектировании. Это в первую очередь зависимость от ряда нефункциональных системных требований, например производительности, защищенности, безопасности, надежности. Предлагаемый подход может быть использован при разработке современного программного обеспечения корпоративных систем управления предприятием, к которым предъявляются повышенные требования по надежности, а также любого другого программного обеспечения.

Серверная часть приложения содержит монолитное крипто-графическое ядро. В первую очередь это обусловлено в простотой реализации. Его недостатком может быть большой набор требований к функциональности приложения, в результате чего будет затруднена поддержка и расширяемость. При добавлении нового функционала программа будет масштабироваться, требуя больше времени и ресурсов на поддержку. Подобный архитектурный прием в системах со слабо связанной архитектурой, каждый компонент ко-

торой обладает минимальными знаниями о других компонентах, позволяет использовать стороннее программное обеспечение. Такая практика позволит во много раз сократить расходы на разработку программного продукта.

Для обеспечения приложения способностью масштабироваться при разработке других слоев, вне ядра, использовался модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных, заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам SOA [7, с. 16 – 17], структура которого изображена на рисунке 2.1.

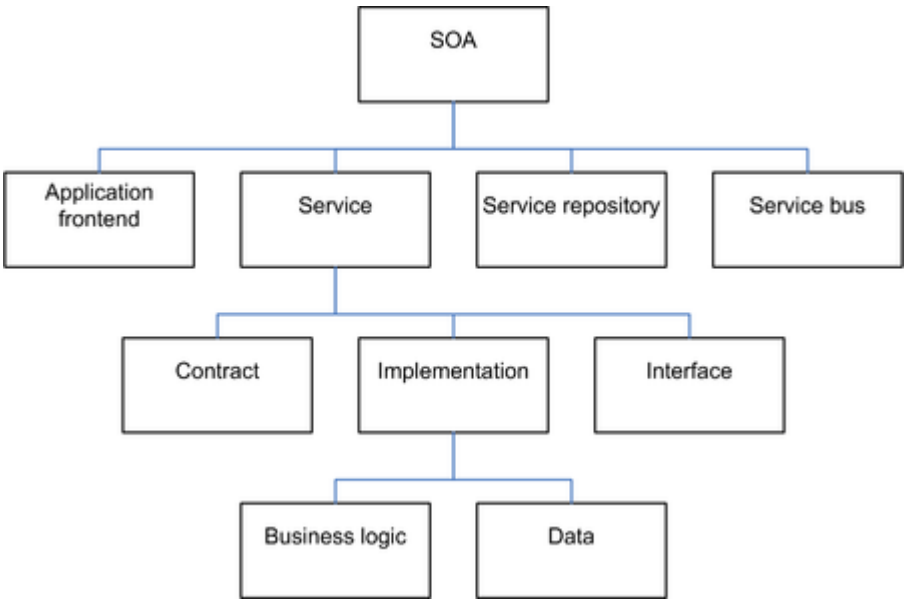

Рисунок 2.1 – Структура протокола SOA.

Программные комплексы, разработанные в соответствии с SOA были реализованы как набор веб-служб, взаимодействующих по протоколу SOAP. Интерфейсы компонентов в сервис-ориентированной архитектуре инкапсулируют детали реализации от остальных компонентов, таким образом, обеспечивая комбинирование и многократное использование компонентов для построения сложных распределённых программных комплексов, обеспечивая независимость от используемых платформ и инструментов разработки, способствуя масштабируемости и управляемости создаваемой системы.

Управление информацией между клиентским и серверным приложением осуществляется по REST протоколу. REST сервис полностью основывается на протоколе передачи данных. Для HTTP действие над данными задается с помощью методов: GET, PUT, POST, DELETE. Таким образом, действия CRUD могут выполняться как со всеми четырьмя методами, так и только с помощью GET и POST запросов [8].

Основные возможности разработанного одностраничного приложения заключаются в написании одного сервиса по выдаче модели отображения и множества сервисов для взаимодействия пользователя с базой данных. Такие возможности заключаются в следующем:

1 Пользователь получает возможность создавать личное хранилище, обладает уникальным идентификатором, который проверяется каждый раз, когда он совершает действия изменяющие состояние сервиса. Если конкретный пользователь пытается просмотреть записи хранилища, которое находится на в шифрованным виде, либо запись, которая принадлежит хранилищу доступного локально, его идентификатор поддается дешифровке и проверяется на валидность в рамках всего цикла жизни клиента.

2 Пользователь может просматривать и редактировать определенные группы данных в хранилищах, дешифрованных его публичным ключом на клиенте, полученным при аутентификации. Добавлять новые поля и группы, имея авторизованный доступ к хранилищу.

3 Пользователю предоставляется возможность добавлять, удалять и редактировать новые поля и хранилища. Создавать новые сертификаты доступа при утере или компрометации последних. Редактировать и восстанавливать доступ к поврежденным сертификатам.

4 При редактировании записей, данные шифруются публичным ключом и отправляются на сервер, где подвергаются дешифровке приватным ключом и вторичному шифрованию синхронным методом.

2.2 Идентификация объектов предметной области

В разрабатываемом приложении можно выделить четыре слоя, связанных между собой различными уровнями обструкции. И хотя у них есть общие части, такие, как модель данных, они вполне могут разрабатываться независимо друг от друга:

– Client Layer;
– Application Server Layer;
– Business Layer;
– Data Layer.

Эта модель объединяет два основных слоя: Client Layer и Application Server Layer. Таким образом, структура программы напоминает классическую трехуровневую модель [7, с. 308 – 401], в которой имеется три явно выраженных слоя. Пользовательский интерфейс, реализованный в Client Layer и Application Server Layer, представляет собой блок отображения данных. Data

Layer приложения, управляемый Business Layer, представляет собой уровень доступа к базе данных. Сюда относятся сервисы, которые поставляются драйвером СУБД, классы-посредники, и синглтоны. База данных представляет уровень Data, самый нижний уровень в иерархии.

Каждый из слоев имеет свой уровень абстракции, согласно которому каждый слой не знает о реализации другого. Все блоки работают по принципу черного ящика, каждому из которых необходимо лишь знать о входных и выходных параметрах. На основании этого можно более детально разобрать каждый элемент системы на подсистемы, поясняющие общий принцип взаимодействия модулей на разных слоях взаимодействия.

Business Layer представляет слой обработки пользовательских данных и событий Data Layer, обеспечивает высокоуровневый интерфейс для Application Server Layer, скрывая реализацию Data Layer. Текущий слой обеспечивает связывание модулей, решает проблему связывания подписчиков и издателей путем автоматической регистрации событий, который автоматически вызывает нужные привязки на основе соглашения об именовании. Существование Business Layer подразумевает регистрацию всех компонентов, которые могут подписываться на события, регистрацию всех событий на которые можно подписаться и набор событий компонентов.

Приемущество разработанной архитектуры связано с отсутствием зависимости между компонентами приложения. Это дает основание для формирования свободной иерархии пакетов, представляя возможность использовать разработанное приложение при построении более сложной архитектуры в качестве стороннего компонента или ядра. Кроме того, благодаря низкому уровню связанности кода, отдельные модули можно также использовать повторно в сторонних продуктах.

2.3 Декомпозиция объектов предметной области

На каждый логически выделенный слой программы возлагаются определенные задачи, решение которых делегируется сущностям более низкого порядка детализации. Кроме того, каждый блок программы так или иначе связан с остальными блоками, чтобы обеспечить работоспособность всего приложения в целом. Связь, как правило, реализуется посредством обмена данными утвержденной сигнатуры между блоками.

Блок пользовательского интерфейса представляет собой множество пользовательских компонентов, взаимодействующими между всеми остальными блоками программы. Он отвечает за отрисовку страницы в браузере.

Основной его функцией является представление того или иного компонента пользователю по запросу от блока контроля и обработки событий. При этом он может обеспечивать рендеринг страницы из разных источников в рамках политики общего происхождения.

Блок контроля и обработки событий определяет наличие изменений состояния блоков пользовательского интерфейса и формирования запросов на сервер, производит оповещение подписчиков о текущем состоянии, служебных сообщениях, не относящиеся напрямую к работе текущего модулю. В первую очередь, это сообщения соединения с сервером, истечение срока действия авторизационный данных, системные ошибки во время исполнения процедур. Блок не имеет представления, как устроено отображение модели, и как работают сервисы, направленные на обновление модели. Блок контроля и обработки событий связан напрямую с блоком пользовательского интерфейса. Все действия и события, происходящие на странице, так или иначе обращаются к контроллеру. Контроллер, после обработки сервисом, возвращает обработанную модель событию на клиентской части, где происходит изменение отображения страницы.

Блок криптографической защиты данных представляет собой совокупность средств и методов шифрования/дешифрования различных типов данных, проверки электронной подписи и сбор данных для их дальнейшей обработки. Осуществляет весь спектор мероприятии по валидации данных и исправлению последствий неавторизованного доступа. Обладает полномочиями по разрыве связи с клиентом при попытке подмены сертификата, возобновление работы после замены старых ключей на новые.

Блок формирования запросов на сервер осуществляет имплементацию данных и их валидацию, а также формирование запросов к серверу соблюдая их иерархию. Все данные, которые отправляются с клиентской части на серверную, проходят проверку в текущем блоке. При соответствии сигнатуры, объекты данных передаются далее на сервер. Все правила, которым должна соответствовать модель, задаются на этапе разработки модели страницы и блоков валидации данных.

Блок серверной обработки HTTP-запросов представляет собой систему маршрутизации по принципу REST архитектуры. Основной причиной для ее внедрении послужило то, что большое количество логики на клиенте необходимо структурировать. К тому же в перспективе решение может масштабироваться. В этом случае возможно появление специальной клиентской части с ограниченным функционалом в качестве мобильного приложения. Не исключено появление и десктопных клиентов приложения. Приложение в ос-

новном использует JSON[1)] для передачи информации и загрузки ее с сервера. Необходимость в максимальной расширяемости системы подразумевает, что к одной и той же точке API сервера могут подключаться разные версии приложения.

Блок обработки пользовательских событий на серверной стороне осуществляет проверку идентификационных данных пользователя, обеспечивая целостность и защиту от фальсификации передаваемой информации. Он может быть использован для обнаружения несанкционированных изменений данных, как преднамеренных, так и случайных, во время жизненного цикла приложения. Любая несанкционированная попытка получить доступ к данным посредствам неавторизованного подключения будет отклонена, а блок известит клиент о попытке несанкционированного доступа.

Блок базы данных представляет собой хранилище всей информации, с которой может работать приложение и пользователь. Блок реализует асинхронную репликацию в конфигурации, основанную на передаче журнала изменений с ведущего узла на ведомые, тем самым поддерживается автоматическое восстановление в случае выхода из строя ведущего узла. Принимает параметры, делает выборку или обновление записей в базе, согласно определенному сценарию, и возвращает набор данных, либо результат выполнения сервису приложения.

Блок объектной модели базы данных и блок обеспечения доступа к базе данных предоставляются драйвером СУБД на уровне платформы. Оба сервиса работают по принципу «черного ящика» – на вход передаются параметры, на выходе возвращаются данные.

[1)] `http://www.w3schools.com/json/`

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Описание конфигурации сборки проекта

В отличие от большинства сборщиков проектов, приложение NPM[1)] использует декларативный способ сборки проектов вместо императивного. Это значит, что JSON файл содержит описание проекта, согласно которому данный проект собирается в исполняемый модуль платформы NodeJS.

В процессе построения проекта происходят следующие этапы:

– синтаксический анализ – проверка на содержание файлом всех необходимых данных и на синтаксическую правильность. В данном пункте проверяется корректность декларативного описанию путем проверки с использованием npmRawCompiler;

– обработка ресурсов и создание основы приложения, загрузка из сети недостающих элементов, описанных в зависимостях;

– компиляция исходных файлов проекта, посредством TypeScriptCompiler и Typings[2)];

– обработка и компиляция тестов;

– тестирование – запуск тестов, только при успешно пройденных тестах выполняется следующий этап;

– формирование исполнительного модуля NodeJS;

Тестовая среда, создаваемая после сборки проекта в исполняемый модуль, имеет все описанные в JSON файле настройки. Файл package.json располагается в корневой директории проекта и является основным источником информации при запуске команд продукта NPM.

Общий вид файла, описывающий целый проект, содержит всю необходимую универсальному декларативному сборщику проектов NPM для формирования исполняемого модуля NodeJS и настройки тестовой среды для модуля расширения согласно описанных в файле настроек. Конфигурация исполнительного модуля приведена в листинге 3.1:

Листинг 3.1 – Конфигурация исполнительного модуля

```
{
  "name": "password-manager",
  "description": "Password-manager of Artem Derevnjuk",
  "version": "0.1.2-rc.3",
  "main": "dist/passman.js",
  "author": "Artem Derevnjuk",
  "repository": {
```

[1)] https://www.npmjs.com/

[2)] https://github.com/typings/typings/

```
  "type": "git",
  "url": "https://github.com/derevnjuk/passman.git"
},
"bugs": {
  "url": "https://github.com/cderevnjuk/passman /issues"
},
"keywords": [
  "passman",
  "password",
  "manager",
  "utility"
],
"dependencies": {
  "body-parser": "^1.15.0",
  "express": "^4.13.4",
  "jsonwebtoken": "^5.7.0",
  "mongoose": "^4.4.8",
  "nconf": "^0.8.4",
  "promise": "^7.1.1",
  "node-uuid": "^1.4.2",
  "clone": "^0.1.19",
  "crypto-js": "^3.1.2",
},
"devDependencies": {
  "chai": "^3.1.0",
  "coveralls": "^2.11.2",
  "es6-promise": "^2.3.0",
  "fs-extra": "^0.26.7",
  "jscs": "^1.13.1",
  "jscs-jsdoc": "^1.3.2",
  "jshint": "~2.8.0",
  "karma": "^0.13.2",
  "karma-browserify": "^4.2.1",
  "karma-firefox-launcher": "^0.1.6",
  "karma-mocha": "^0.2.0",
  "karma-mocha-reporter": "^1.0.2",
  "mocha": "^2.2.5",
  "native-promise-only": "^0.8.0-a",
  "nodeunit": ">0.0.0",
  "nyc": "^2.1.0",
  "recursive-readdir": "^1.3.0",
  "rimraf": "^2.5.0",
  "rollup": "^0.25.0",
  "rollup-plugin-node-resolve": "^1.5.0",
  "rollup-plugin-npm": "~1.3.0",
  "rsvp": "^3.0.18",
  "semver": "^4.3.6"
},
"scripts": {
  "coverage": "nyc npm test && nyc report",
  "coveralls": "nyc npm test && nyc report --reporter=text-lcov | coveralls",
  "lint": "jshint lib/ test/ mocha_test/ perf/memory.js perf/suites.js perf/
      benchmark.js support/ karma.conf.js && jscs lib/ test/ mocha_test/ perf/memory.js
```

```
        perf/suites.js perf/benchmark.js support/ karma.conf.js",
    "mocha-browser-test": "karma start",
    "mocha-node-test": "mocha mocha_test/ --compilers js:babel-core/register",
    "mocha-test": "npm run mocha-node-test && npm run mocha-browser-test",
    "nodeunit-test": "nodeunit test/test-async.js",
    "test": "npm run-script lint && npm run nodeunit-test && npm run mocha-node-test"
  },
  "license": "MIT",
  "jam": {
    "main": "dist/passman.js",
    "include": [
      "dist/passman.js",
      "README.md",
      "LICENSE"
    ],
    "categories": [
      "Utilities"
    ]
  },
  "spm": {
    "main": "dist/passman.js"
  },
  "volo": {
    "main": "dist/passman.js",
    "ignore": [
      "**/.*",
      "node_modules",
      "bower_components",
      "test",
      "tests"
    ]
  }
}
```

Данный файл для сборщика NPM преобразуется неявно в совокупность свойств. Описание основных свойств файла программной модели описано ниже:

– name, значение определяет уникальное имя пакета, пространство имен, общее для всех единиц исполняемого модуля;

– description, описание пакета, предоставляет исчерпывающую информацию по исполняемому модулю при его публикации;

– version, версия исполняемого модуля;

– main, точка входа в приложение, в тоже время идентификационное имя исполняемого модуля;

– author, информация об авторе и поставщике исполняемого модуля для платформы. Содержит такую информацию, как имя автора и ссылка на более подробную информацию о разработчике модуля;

– repository, публичный адрес на репозиторий в системе контроля вер-

сий, описывающий порядок доступа к проекту приложения;

– bugs, связывает bugs-tracking с системами автоматизированного обслуживания и отчетности;

– keywords, список ключевых слов для поиска пакета в менеджере пакетов;

– dependencies, список зависимостей, необходимых для корректного запуска приложения, набор описаний зависимых компонент модуля, описываются загружаемые в момент сборки компоненты, обеспечивающих основу модуля при сборке.

– devDependencies, список параметров пакетов рабочего окружения, который определяет набор исполняемых пакетов сборщик NPM, GULP и SystemJS, необходимых для успешной сборки исполняемого модуля;

– scripts, словарь, содержащий скриптовые команды, которые запускаются во время жизни приложения по установленному условию;

– jam, список клиентских зависимостей, динамически внедряемых пакетов, необходимых для успешного запуска клиентского приложения.

– spm, статический идентификатор в системе пакетов SPM.

– volo, конфигурация конструктора модулей AMD, CommonJS, Node.

В данном исполняемом модуле используются следующие компоненты и зависимости, описанные в dependencies:

– body-parser – межпрограммный модуль обработки тела запроса, ответственный за разбор POST запросов от клиента;

– express – реактивный северный-фраймворк для NodeJS, предоставляющий основные элементы взаимодействия с функциональностью на стороне клиента посредством протокола SOAP. Содержит основной набор классов, необходимый для реализации REST-сервиса;

– jsonwebtoken – компонент, обеспечивающий аутентификацию по методу JSON Web Token[1)] со средствами хеширования JWT маркера;

– mongoose – система, которая предоставляет возможность объектно-реляционного отображения объектной модели приложения, в соответствии с интерфейсом NodeJS.

– nconf – основной компонент динамической конфигурации сервера на базе объектной модели. Является наиболее часто используемым компонентов взаимодействия сервера с нижестоящими слоями;

– promise – компонент спецификации ES2016[2)], предоставляющий одни из рекомендуемых способов организации асинхронного кода

[1)] `https://jwt.io/`

[2)] `https://github.com/tc39/ecma262`

– node-uuid — модуль авторизации, поддерживающий JWT маркеры.

– clone — стандартный компонент многопроцессорного взаимодействия, производящий обмен данных между объектной моделью приложения и базой данных

– crypto-js — библиотека содержит реализацию функций шифрования и хеширования на северной и клиентской стороне.

В файл программной модели модуля могут также быть добавлены дополнительные свойства. При выполнении консольной команды npm run происходит полный цикл построение программного модуля, запуск тестов и запуск сервера NodeJS для работы тестовой среды приложения. На программный модуль при использовании сборщика NPM накладывается ряд ограничений. Запуск команды npm run приводит к созданию в корневом каталоге папок node_modules и typings, содержимое которых заполняется по мере выполнения команды. После успешного выполнения в папке node_modules можно найти зависимости приложения и модуля тестирования данного программного средства, а также файлы тестовой среды для исполнительного модуля. Которые представляют собой одиночную версию разрабатываемого продукта в режиме быстрой разработки. Данный режим сканирует любые изменения исходного кода модуля и изменяет по мере выполнения функциональность определенных в модуле элементов. Эта возможность позволяет не перезапускать команду лишний раз при наличии изменений.

3.2 Классы Business Layer

К классам Business Layer относятся классы, наследующие от класса EventEmittor, который находится в пространстве имен NodeJS.events.EventEmittor. Основной ролью данных классов является формирования логики обработки всех декларированных объектов объектной модели во время жизненного цикла приложения в рамках асинхронной модели вводавывода. Все классы принадлежащие этому слою являются реализацией интерфейса IBusinessSecurityData.

3.2.1 Класс ObserverServer является базовым абстрактным классом для классов бизнес логики и наследует свойства и методы абстрактного класса EventEmittor. Класс не содержит дополнительных атрибутов и модификаторов доступа.

Поля:

– _logger – поле типа Logger, используется для логирования исключительных ситуаций, сбоев и отсутствие разрешений;

– applicationProperties – поле типа ApplicationProperties, позволяющее получить свойства текущей конфигурации ObserverServer;

– _dataResultFactory – поле типа IDataResult, возвращающее выборку фабрики результатов, формирует модель результат валидации для дальнейшей передачи на слой выше.

Методы:

– IObserverServer ObserverServer(ApplicationProperties applicationProperties) – конструктор класса, принимающий в качестве параметров объект

– конфигурации;

– boolean _hasAdminPermission() – метод, определяющий наличие прав на администрирование пользователей и групп. Метод используется для выполнения запросов к базе данных, предотвращая несанкционированные попытки получить пользовательские данные посредством не авторизованного доступа.

3.2.2 Класс Westley представляет собой класс, реализующий функционал редактора хранилища и менеджер истории его изменений. Является реализацией интерфейса IWestley.

Поля:

– _dataset – поле типа IStorageData исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _history – поле типа Array<IStorageHistory> контейнер истории для текущего пользователя;

– _cachedCommands – поле типа Map<Command>, содержащая результаты наиболее частых запросов к хранилищу.

Методы:

– IWestley clear() – метод возвращающий пустой объект хранилища.

– IWestley execute(Command command) – метод, принимает команды типа Command и делегирует их выполнение объекту command-Tools типа ICommandTools, сохраняя историю и изменяя _dataset.

– Command _getCommadForKey(string commandKey) – метод возвращает command типа Command по переданному ключу из свойства _cachedCommands.

– IWestley pad() – метод возвращающий хранилище, внедряя смещение в оригинальный прототип хранилища.

– IStorageData getDataset() – метод возвращает результат выполнения процедуры в базе данных.

– Array<IStorageHistory> getHistory() – метод возвращает контейнер истории запросов к хранилищу.

3.2.3 Класс Archive управляет временем существования компоновкой и обработкой объектов ManagedEntry и ManagedGroup. Данный класс может иметь только один экземпляр для конкретного пользователя.

Поля:

– _westley – поле типа IWestley, инкапсулирует редактор хранилища и менеджер истории запросов к нему;

– date – поле типа Date, содержит дату создания экземпляра класса Archive.

Методы:

– Archive() – конструктор класса;

– Array<ManagedEntry> findEntriesByCheck(Achive achive, string check, string key, RegExp value) – статический метод, возвращающий массив всех записей экземпляра класса Archive на основании переданных мета-значений;

– boolean containsGroupWithTitle(string groupTitle) – метод сопоставления всех существующих групп переданному заголовку группы;

– IManagedGroup createGroup(string title) – метод принимает имя группы в качестве аргумента и возвращает ее имплементацию;

– Array<IManagedGroup> findEntriesByMeta(string metaName, RegExp value) – метод сбора релевантной выборки групп по переданным, в качестве параметров, мета-данным и их значению типа string, и RegExp соответственно;

– Array<IManagedGroup> findEntriesByProperty(string property, RegExp value) – метод сбора релевантной выборки групп по переданным, в качестве параметров, свойству и его значению типа string, и RegExp соответственно;

– Array<IManagedGroup> findGroupsByTitle(string title) – метод поиска всех групп архива, которые соответствуют преданному имени title;

– IManagedEntry getEntryByID(string entryID) – метод, осуществляющий поиск записи по ее уникальному идентификатору;

– IManagedGroup getGroupByID(string groupID) – метод, который производит рекурсивный поиск по идентификатору группы;

– Array<ManagedGroup> getGroups() – метод возвращает все группы, которые содержит экземпляр класса Archive;

– ManagedGroup getTrashGroup() – метод, позволяющий получить удавленные группы, подписанные на экземпляр класса Archive;

– IWestley _getWestley() – метод возвращает базовый экземпляр типа

IWestley;

– Achive createWithDefaults() – метод, возвращающий экземпляр класса Archive с параметрами по умолчанию.

3.2.4 Класс InigoCommand представляет собой класс для обработки и выполнения команд типа Command, наследующий от статического класса REPL.

Поля:

– _commandKey – поле типа string, которое содержит индекс выполняемой команды;

– _commnadArgs – поле типа Array<strign arg> вмещает набор аргументов команды;

– commandArgument – статическое поле типа Map<ItemMetaRoot>, хранит маркеры и идентификаторы, присвоенные Command по умолчанию;

– commands – статическое поле типа Map<Command>, описывающее все подписанные команды в рамках текущей реализации CommandArguments.

Методы:

– InigoCommand() – конструктор класса;

– InigoCommand addArgument(...args) – метод принимает неизвестное количество аргументов, привязывая их к текущей выполняемой команде, возвращает

– объект типа InigoCommand;

– Command generateCommand() – метод возвращающий реализацию команды на основании полей _commandKey и _commnadArgs;

– InigoCommand create(string cmd) – статический метод, возвращающий один и только один экземпляр класса InigoCommand для единственной команды по переданному индексу.

3.2.5 Класс ManagedGroup представляет собой класс для управления группами записей пользователя, наследующий от стандартного класса Stream. Определяет порядок привязки групп к хранилищам пользователя и подписку на события глобальной группы. Является реализацией интерфейса IManagedGroup.

Поля:

– _archive – поле типа string, которое содержит индекс выполняемой команды;

– _westley – поле типа Array<string> вмещает набор аргументов команды;

– _remoteObject – поле типа IremoteObject, содержащее ссылку на удаленную группу подписанную на события экземпляр ManagedGroup.

Методы:

– ManagedGroup() – конструктор класса;

– ManagedEntry createEntry(string title) – метод создания новой записи с заголовком title, возвращающий экземпляр класса ManagedEntry;

– ManagedGroup createGroup(string title) – метод создания дочерний группы с именем title, возвращающий экземпляр класса;

– void delete() – метод удаления группы типа ManagedGroup и очистки _westley и _removeObject;

– ManagedGroup deleteAttribute(string attr) – метод удаления атрибута по переданному имени attr, возвращает ссылку на экземпляр класса

– ManagedGroup;

– string getAttribute(string attributeName) – метод принимает имя атрибута и возвращает значение релевантного атрибута типа string;

– Array<ManagedEntry> getEntries() – метод, возвращающий массив записей группы типа ManagedEntry, привязанной к экземпляру класса;

– Array<ManagedGroup> getGroup() – метод, возвращающий массив всех групп типа ManagedGroup привязанных к глобально группе;

– string getID() – метод возвращает идентификатор глобальной группы;

– string getTitle() – метод возвращает имя глобальной группы;

– boolean isTrash() – метод, осуществляющий проверку на тип группы, возвращает значение boolean;

– ManagedGroup moveToGroup(ManagedGroup group) – метод перемещения подчиненной группы в группу group, переданную в качестве параметра, возвращает ссылку на экземпляр класса;

– ManagedGroup setTitle(string title) – метод, который принимая заголовок группы типа string, устанавливает ее в качестве имени глобальной группы;

– ManagedGroup setAttribute(string attributeName, string value) – метод принимает имя свойства и его значения, присваивая его глобальной группе;

– IArchive _getArchive() – метод возвращает реализацию интерфейса IArchive – владельца глобальной группы;

– IRemoteObject _getRemoteObject() – метод, который возвращает значение поля remoteObject текущего экземпляра класса;

– IWestley _getWestley() – метод, который возвращает значение поля _westley в контексте экземпляра класса;

– ManagedGroup createNew(IArchive archive, string parentID) – статический метод, позволяющий создать новый экземпляр класса ManagedGroup в хранилище archive с ведущей группой, обладающей parentID.

3.2.6 Класс ManagedEntry используется для инициализации записи пользователя. Наследует свойства и методы абстрактного стандартного класса BaseManagementEntry. Имеет шаблон _remoteObject для представления и инициализации функциональности на стороне клиента.

Поля:

– _archive – поле типа string, которое содержит индекс выполняемой команды;

– _westley – поле типа Array<string> вмещает набор аргументов команды;

– _remoteObject – поле типа IRemoteObject, содержащее ссылку на скрытый объект-токен, содержащий хранимую информацию.

Методы:

– ManagedEntry(Archive archive, IRemoteObject remoteObj) – конструктор класса, в качестве параметров принимает экземпляр archive и ссылку на удаленное представление remoteObj;

– void delete() – метод удаляет запись, в случае, если запись уже удалена, полностью очищает все ссылки на нее;

– ManagedEntry deleteAttribute(string attr) – метод, удаляющий параметр записи, переданный в качестве единственного аргумента attr. Возвращает ссылку на текущий экземпляр записи типа ManagedEntry;

– ManagedEntry deleteMeta(string property) – метод удаляет метаданные по переданному параметру property. Возвращает ссылку на экземпляр на котором был вызван;

– string getAttribute(string attr) – метод возвращает значение параметра записи по преданному имени атрибута;

– DisplayInfo getDisplayInfo() – метод возвращает экземпляр класса DisplayInfo, который представляя объект типа JSON;

– MangedGroup getGroup() – метод возвращает ссылку на группу типы ManagedGroup, содержащий экземпляр текущей записи;

– string getMeta(string property) – метод возвращает мета-данные по ключу property;

– string getID() – метод, возвращающий идентификатор записи в качестве строки;

– string getProperty(string property) – метод возвращает значение свой-

ства, переданного в качестве первого аргумента;

– ManagedEntry moveToGroup(ManagedGroup group) – метод позволяет переместить запись в группу, ссылку на которую принимает в качестве первого параметра;

– ManagedEntry setAttribute(string name, string value) – метод присваивает атрибут с именем name и значением value записи, возвращая ссылку на нее;

– ManagedEntry setProperty(string name, string value) – метод присваивает свойство с именем name и значением value записи, возвращая ссылку на нее;

– IArchive _getArchive() – метод возвращает реализацию интерфейса IArchive, содержащего все записи и группы;

– IRemoteObject _getRemoteObject() – метод, который возвращает значение поля remoteObject текущего экземпляра класса;

– IWestley _getWestley() – метод, который возвращает значение поля _westley в контексте экземпляра класса;

– ManagedEntry createNew(IArchive archive, string parentID) – статический метод, позволяющий создать новый экземпляр класса ManagedGroup в хранилище archive с ведущей группой, обладающей parentID.

3.2.7 Класс Credentials используется для формирования и проверки прав доступа к хранилищам. Наследует свойства и методы абстрактного стандартного класса BaseManagementEntry.

Поля:

– _signing – поле типа Array<string> вмещает набор аргументов команды;

– _model – поле типа IRemoteObject, содержащее ссылку на скрытый объект-токен, содержащий хранимую информацию.

Методы:

– Credentials(Model data) – конструктор класса, в качестве единственного параметра принимает экземпляр типа Model;

– Credentials setIdentity(string username, string password) – метод устанавливает username и password поля _model текущего экземпляра. Возвращает экземпляр класса Credentials;

– Credentials setType (CredentialsType type) – метод присваивает полномочия типа type, переданного в качестве первого параметра, модели поля _model;

– Promise<IOcane> convertToSecureContent(string password) – метод пре-

образующий учетные данные в шифрованную строку посредством токена password. Возвращает Promise типа IOcane;

– Promise<IOcane> createFromSecureContent(IOcane content, password) – статический метод, возвращающий экземпляр класса Credentials, содержащий дешифрованные данные content.

3.2.8 Класс Certificate используется для формирования и проверки подписи сертификата. Инкапсулирует логику по оптимизации вычислений больших простых чисел. Наследует свойства и методы абстрактного стандартного класса BaseManagementEntry.

Поля:

– _GCM – поле типа Buffer, содержит дополнительную информацию, используемую в качестве дополнительно параметра при проверки подлинности подписи;

– _generator – поле типа IdiffieHellman, вмещает ссылку на генератор Диффи-Хеллмана в кодировке поля _GCM;

– digest – поле типа Int32Array, словарь описывающий форму хеширования данных для экземпляров класса EncodeForfatter;

– _level – поле типа number, отражает количество итераций сдвига случайных кривых;

– _timestamp – поле типа Date, содержит расчетное значение времени необходимого для осуществления шифрования;

– _crypto – поле типа ICrypto, ссылка на композицию функций шифрования и хеширования.

Методы:

– Certificate() – конструктор класса;

– Buffer exportChallenge(ISuperToken spkac) – метод, принимает аргумент типа ISuperToken, который вмещает в себя открытый ключ и проверку типа Challenge. Возвращает объект типа Buffer, инкапсулирующий переданный ключ и тайну;

– boolean verifySpkac(Buffer spkac) – метод осуществляет проверку на соответствие тайны и публичного ключа;

– Buffer getAuthTag() – метод возвращает тег аутентификации типа Buffer, прошедшего проверку подлинности шифрования;

– Buffer computeSecret(ISuperToken spkac) – метод принимает аргумент типа ISuperToken, на основании которого производит вычисление секрета по протоколу Диффи-Хеллмана. Возвращает секрет типа Buffer;

– IDiffieHellman getGenerator() – метод возвращает _generator типа IDif-

fieHellman;

– Buffer getPrime() – метод возвращает большое простое число типа Buffer;

– Buffer getPrivateKey() – метод возвращает приватный ключ типа Buffer;

– Buffer getPublicKey() – метод возвращает публичный ключ типа Buffer;

– void mac(Uint8Array message, Int32Array hmac) – метод, осуществляющий проверку подлинности хеш-функции hmac посредством переданного ключа message типа UInt8Array;

– Int32Array createMAC(Buffer password) – метод, позволяющий создать хеш-функцию для переданного пароля password типа Buffer;

– Buffer keyBlock(Uint8Array password, Uint8Array salt, number iteration, number blockIndex) – метод вычисления блока ключа для переданной соли типа Uint8Array и пароля password типа Uint8Array. Возвращает blockIndex блок ключа на итерации iteration, номер которой передан третьим параметром;

– Buffer compress(string data) – метод осуществляет декрементальный разбор данных и их сжатие по словарю. Возвращает сжатые данные типа Buffer;

– string decompress(Buffer data) – метод производит восстановление сжатых данных по словарю с их последующей нормализацией и смещением преобразующей формы с относительным смещением в форму с абсолютным;

– string normalizTextValue(string data) – метод нормализует данные после восстановления. Возвращает строку типа string;

– string getUniqueID() – метод возвращает индикатор среды распределенных вычислений, определяя уникальный флаг хранилища пользователя;

– string hashText(string data) – метод, осуществляющий хеширование данных типа string, переданных ему в качестве перового параметра;

– Buffer decideImpl(number iteration, number blockIndex) – метод нормализует секрет Диффи-Хеллмана. Возвращает секрет типа Buffer;

– Buffer encrypt(Buffer data, Buffer key) – метод производит шифрование сжатых данных data по ключу key;

– Buffer decrypt(Buffer ciphertext, Buffer pubkey) – метод производит дешифрование сжатых данных ciphertext по публичному ключу pubkey.

– number _nbi() – метод возвращает число типа BigInteger.

3.3 Классы Data Layer

К классам Business Layer относятся классы, наследующие от класса EventEmittor, который находится в пространстве имен NodeJS.events.Event-

Emittor. Основной ролью данных классов является формирования логики обработки всех декларированных объектов объектной модели во время жизненного цикла приложения в рамках асинхронной модели вводавывода.

3.3.1 Класс AddGeneralDataAction представляет собой класс, реализующий добавление новых записей в хранилище. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя.

Методы:

– AddGeneralDataAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.2 Класс AddMiscItemDataAction представляет собой класс, реализующий добавление новых записей в таблицу Miscellaneous хранилища. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– AddMiskItemDataAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.3 Класс AddPartComponent представляет собой класс, реализующий добавление новых записей типа KeyComponent. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– AddPartComponent() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.4 Класс AddPaymentsRoutingDataAction представляет собой класс, реализующий добавление новых записей типа PaymentsRouting. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– AddPaymentsRoutingDataAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.5 Класс AddSubComponentDataAction представляет собой класс, реализующий добавление новых полей типа SubKeyComponent. При вызове данный класс строит объект JSON, в котором содержатся поля, необходимые для добавления новой записи. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– AddSubKeyComponentDataAction () – конструктор класса;

– Task<IDataResult<PartCostDetailFormModel, PartCostDetailFormModel» Pro-cessAsync(PartCostDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.6 Класс AddSupplyItemDataAction представляет собой класс, реализующий добавление новых полей типа SupplyItemsKey. При вызове данный класс строит объект JSON, в котором содержатся поля, необходимые для добавления новой записи. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partCostDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartCostDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– AddSupplyItemDataAction() – конструктор класса;

– Task<IDataResult<PartCostDetailFormModel, PartCostDetailFormModel» Pro-cessAsync(PartCostDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.7 Класс CalculateRightSummaryTotalstDataAction представляет собой класс, реализующий пересчет итогового значения прав на все группы, со-

ставляющих его хранилище. Является реализацией интерфейса IDataAc-tion.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры. Является экземпляром сервиса доступа к базе данных. Данное поле инициализируется на этапе запуска приложения с помощью контейнера;

– _partCostDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartCostDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– CalculateRightSummaryTotalstDataAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.8 Класс CalculateProcessDataAction представляет собой класс, реализующий учет действий пользователя, который был сконфигурирован пользователем. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partCostDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartCostDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– CalculateProcessDataAction () – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.9 Класс CalculateAccessesUserDataAction представляет собой класс, реализующий оценку прав пользователя на доступ к хранилищам. Является реализацией интерфейса IDataAction.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partCostDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartCostDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– CalculateProcessDataAction () – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий обработку данных и запись этих данных в таблицу. На вход поступает модель, которую необходимо обновить, на выходе получается результат добавления.

3.3.10 Класс CheckProcessRoutingDeleteDataAction представляет собой класс, который проверяет возможность удаления записей с хранилища. Если процесс удалить не возможно, метод класса вернет валидационную ошибку с текстом сообщения. Иначе вернется результат успешной проверки, и запись будет удалена из базы данных. Является реализацией интерфейса IData-

Action.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

– _updatePartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных.

Методы:

– CheckProcessRoutingDeleteDataAction () – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий проверку данных на наличие связей в базе данных. На вход поступает модель, которую необходимо проверить, на выходе получается валидационный результат;

3.3.11 Класс GetPartDetailDataAction представляет собой класс, который позволяет получить данные для модели на клиенте. Является реализацией интерфейса IDataAction.

Поля:

– _dataResultFactory – поле фабрики результатов, формирует модель результат валидации для дальнейшей передачи на клиент;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант;

– _getPartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных для получения модели.

Методы:

– GetPartDetailDataAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, выполняющий получающий данные для конкретной записи в базе данных. На вход поступает модель с ключами, на выходе получается модель типа Model.

3.3.12 Класс ViewPartDetailFormAction представляет собой главный класс, так называемую точку входа в приложение. Данный класс является фабрикой по формированию и наполнению модели данных на серверной стороне приложения. В приложении так же используются однотипные классы для формирования объектов отображения. Подобные классы являются реализацией интерфейса IViewAction. Единственным отличием между ManagedGroup и ManagedEntry является тип возвращаемой модели. Для ManagedGroup это ManagedGroupModel, для ManagedEntry – ManagedEntryModel.

Поля:

– _actionBarModelBuilderFactory – поле фабрики панели действий, формирует модель действий для дальнейшей передачи на клиент;

– _partDetailQuoteHelper – поле класса помощника, выполняющего проверку на разрешение редактирования записи, и контейнера констант, выполняет роль контейнера, содержащего в себе текстовые константы и методы, которые необходимы для получения разрешений и настроек;

– _getPartDetailFormDataHelper – помощник для работы с сервисом запросов базы данных для получения модели.

– _ModelBuilder* – генератор классов-фабрик, которые возвращают конкретную модель для наполнения формы. В данном приложении используется два типа моделей: модели таблицы и модель формы, которая выступает контейнером для таблиц. Все такие классы принимают на вход модель данных, возвращают реализацию интерфейса ISectionViewModelBuilder.

Методы:

– ViewPartDetailFormAction() – конструктор класса;

– Task<IDataResult<PartDetailFormModel, PartDetailFormModel» ProcessAsync(PartDetailFormModel contextModel) – метод, получающий данные для конкретной записи в базе данных, формирующий из этих данных контекстную модель, инициирующий модель отображения и наполняющий модель отображения секциями. На вход поступает модель с ключами, на выходе получается модель отображения;

3.3.13 Класс GetPartDetailFormDataHelper представляет собой класс, возвращающий данные хранилища по запросу пользователя.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения

процедуры;

– _glossaryWordProvider – поле, позволяющее получить глосаризованное значение текстовых составляющих записей;

– _quoteWizardSettings – контейнер настроек для текущего пользователя;

Методы:

– GetPartDetailFormDataHelper () – конструктор класса;

– Task CalculateAccessUserModel contextModel) – метод калькуляции прав доступа к хранилищу;

– Task GetModelMarkupBreakdownData (Model contextModel) – метод получения данных для групп хранилища пользователя;

– Task GetLinkedInlineProcessRouting (Model contextModel) – метод метод получения данных для таблицы встроенных процессов;

– Task GetMarkupSummary (Model contextModel) – метод возвращающий реализацию объекта типа IMarkupData;

– Task GetPartDetailFormModelData (Model contextModel) – метод получения данных для формы, обладающий наибольшим приоритетом, вызывается в первую очередь;

– Task GetQuoteRaw(Model contextModel) – метод получения данных для таблицы с побочными полями текущей записи;

– Task CalculateMarkup (Model contextModel) – метод валидации, возвращающий оценочную стойкость пароля;

– Task GetGlossaryLabels (Model contextModel) – метод получения локализованных строковых литералов;

– Task GetPartDetailItems (Model contextModel) – метод получения данных для каждой записи текущей группы;

– Task GetPasswordRouting (Model contextModel) – метод получения паролей для конкретной записи;

– Task GetQuotePartComponents (Model contextModel) – метод составляющих компоненты в производственном цикле, вне виртуального событийного цикла слоя;

– Task SetEntrySummary(Model contextModel) – метод установки новой записи;

3.3.14 Класс UpdatePartDetailFormDataHelper представляет собой класс-сервис, обновляющий данные для страницы. Класс аналогичен предыдущему 3.3.14, за исключением того, что данные отправляются в базу, а не считываются.

Поля:

– _applicationContext – поле контекста приложения, содержит в себе информацию о текущем пользователе, локализации;

– _dataSourceInvoker – поле исполнения процедуры в базе данных. На вход принимает модель запроса, на выходе возвращает результат выполнения процедуры;

– _glossaryWordProvider – поле, позволяющее получить глосаризованное значение текстовых составляющих страницы;

– _quoteWizardSettings – контейнер настроек текущего пользователя.

Методы:

– UpdatePartDetailFormDataHelper() – конструктор класса;

– Task UpdateModelMarkupBreakdownData (Model contextModel) – метод обновления данных для групп хранилища пользователя;

– Task UpdateLinkedInlineProcessRouting (Model contextModel) – метод метод обновления данных для таблицы встроенных процессов;

– Task UpdateMarkupSummary (Model contextModel) – метод обновления реализации объекта типа IMarkupData;

– Task UpdatePartDetailFormModelData (Model contextModel) – метод обновления данных для формы, метод получения данных, обладающий наибольшим приоритетом, вызывается в первую очередь;

– Task UpdateQuoteRaw(Model contextModel) – метод обновления данных для таблицы с побочными полями текущей записи;

– Task UpdatePartDetailItems (Model contextModel) – метод обновления данных для каждой записи текущей группы;

– Task UpdatePasswordRouting (Model contextModel) – метод обновления процессов для конкретной записи;

– Task UpdateQuotePartComponents(Model contextModel) – метод обновления составляющих и компонент производственного процесса странице;

3.4 Классы Application Server Layer

К классам и методам сервисов относятся классы и методы, имеющие атрибуты @Path и принимающих в качестве параметра объект типа Request. Основной ролью данных классов и методов является формирования ответов для пользователя с целью дальнейшего взаимодействия через сервисы посредством AJAX. Методы сервиса являются основными методами влияния на среду выполнения приложения.

3.4.1 Класс UserManagementRestResource используется для методов сервиса по управлению пользователями и группами. Добавлен атрибут @Path со значением /users. Является реализацией интерфейса IUserManage-mentService.

Поля:

– userUtil – поле типа UserUtil, хранящее менеджер пользователей приложения;

– groupManager – поле типа GroupManager, используется для логирования исключительных ситуаций, сбоев и отсутствие разрешений;

– logger – поле типа Logger, используется для логирования исключительных ситуаций, сбоев и отсутствие разрешений.

Методы:

– UserManagementRestResource (UserUtil userUtil, GroupManager groupManager) – конструктор класса сервиса по управлению пользователями и группами, принимающего в качестве менеджер пользователей и менеджер групп;

– Response addUserToGroup(Request req) – метод сервиса, который добавляет пользователя к группе. Метод при успешном выполнении возвращает значение в формате JSON {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /addToGroup. Идентификаторы групп и пользователя передаются в качестве параметров HTTP запроса;

– Response moveUserFromGroup(Request req) – метод сервиса, который перемещает пользователя из одной группы в другую. Метод при успешном выполнении возвращает {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /moveFromGroup. Идентификаторы групп и пользователя передаются в качестве параметров HTTP запроса;

– Response removeFromGroup(Request req) – метод сервиса, который удаляет пользователя из группы. Метод при успешном выполнении возвращает значение в формате JSON {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /addToGroup. Идентификаторы групп и пользователя передаются в качестве параметров HTTP запроса;

– Response getUserInfo(Request req) – метод сервиса, который возвращает данные о конкретном пользователе. Метод при успешном выполнении возвращает сериализованный объект User в формате JSON . Атрибут @Path имеет значение /addToGroup. Идентификатор пользователя передаются в качестве параметра HTTP запроса.

3.4.2 Класс LabelManagementRestResource используется для методов сервиса по управлению записями, группами и хранилищами. Добавлен атрибут @Path со значением /labels. Является реализацией интерфейса ILabel-

ManagementService.

Поля:

– labelManager – поле типа LabelManager, хранящее менеджер записей приложения;

– issueManager – поле типа IssueManager, используется для логирования исключительных ситуаций, сбоев и отсутствие разрешений;

– logger – поле типа Logger, используется для логирования исключительных ситуаций, сбоев и отсутствие разрешений.

Методы:

– LabelManagementAction(LabelManager labelManager, IssueManager issueManager) – конструктор класса сервиса по управлению записями, группами и хранилищами, менеджер записей и задач;

– Response addIssueToLabel(HttpServletRequest req) – метод сервиса, который добавляет группу к хранилищу. Метод при успешном выполнении возвращает значение в формате {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /addToLabel. Идентификатор задачи и значение метки передаются в качестве параметров HTTP запроса;

– Response moveIssueFromLabel(Request req) – метод сервиса, который удаляет записи с одной группы и добавляет к другой. Метод при успешном выполнении возвращает {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /moveFromLabel. Идентификатор задачи и значение метки передаются в качестве параметров HTTP запроса;

– Response removeIssueFromLabel(Request req) – метод сервиса, который удаляет запись из группы. Метод при успешном выполнении возвращает значение в формате JSON {"responseType": "Success", "message": ""}. Атрибут @Path имеет значение /removeFromLabel. Идентификатор задачи и значение метки передаются в качестве параметров HTTP запрос.

3.4.3 Класс BaseResponse используются для взаимодействия с функциональностью клиента, сервисов и серверных элементов функциональных действий. Класс BaseResponse является базовым классом для любой сущности типа Application Server Layer.

Поля:

– responseType – поле типа ResponseType, которое хранит статус ответа;

– message – поле строкового типа, которое используется для передачи сообщения об ошибке или для оповещения;

Методы:

– BaseResponse() – конструктор класса базового ответа сервиса и функционального действия;

– BaseResponse(ResponseType responseType, String message) – перегрузка конструктора класса базового ответа сервиса и функционального действия с параметрами;

– getMessage() и setMessage(String message) – методы для получения и установки значения сообщения;

– getResponseType() и setResponseType(ResponseType responseType) – методы для получения и установки типа ответа сервиса и функционального действия.

3.4.4 Класс Quickmanagement является классом связи с сервисом и определяет события подписки на запросы пользователя.

Методы:

– Quickmanagement addIssueToLabel(Array<string> ...options, Function callback) – метод асинхронного вызова метода addIssueToLabel сервиса LabelManagementService. Параметр options должен содержать идентификатор записи, идентификатор группы. В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement moveIssueFromLabel(Array<string> ...options, Function callback) – метод асинхронного вызова метода moveIssueFromLabel сервиса LabelManagementService. Параметр options должен содержать идентификатор задачи, идентификаторы метки назначения и метки источника. В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement removeIssueFromLabel(Array<string> ...options, Function callback) – метод асинхронного вызова метода removeIssueFromLabel сервиса LabelManagementService. Параметр options должен содержать идентификатор задачи, идентификаторы метки, из которой удаляется задача. В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement addUserToGroup(Array<string> ...options, Function callback) – метод асинхронного вызова метода addUserToGroup сервиса UserManagementService. Параметр options должен содержать идентификатор пользователя, идентификатор группы. В качестве значения параметра callback пе-

редается функция, которая будет вызвана после возврата ответа сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement moveUserFromGroup(Array<string> ...options, Function callback) – метод асинхронного вызова метода moveUserFromGroup сервиса UserManagementService. Параметр options должен содержать идентификатор пользователя, идентификаторы группы назначения и группы источника. В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement removeFromGroup(Array<string> ...options, Function callback) – метод асинхронного вызова метода removeFromGroup сервиса UserManagementService. Параметр options должен содержать идентификатор пользователя, идентификатор пользователя, который удаляется из группы. В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа от сервиса. Функция callback принимает в качестве параметра объект класса BaseResponse;

– Quickmanagement getUserInfo(Array<string> ...options, Function callback) – метод асинхронного вызова метода getUserInfo сервиса UserManagementService. Параметр options должен содержать идентификатор пользователя В качестве значения параметра callback передается функция, которая будет вызвана после возврата ответа от сервиса. Функция callback принимает в качестве параметра объект класса User;

– T parseResponse(Response response) – метод обработки ответа сервиса. Параметр response имеет значение ответа сервиса.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

4.1 Алгоритм расчета апостериорной вероятности для оценки стойкости шифрования

В данном подразделе рассматривается известный алгоритм, использующий оценку апостериорной вероятности в качестве критерия поиска стойкой тайны для шифрования данных. Подробное описание данной оценки и базового алгоритма поиска приведены в работе [6].

Существуют подходы, использующие байесов метод для оценки качества полученной тайны и алгоритмы на их базе пытаются максимизировать апостериорную вероятность структуры шифра для данного набора экспериментальны данных. В программном обеспечении, разработанном в данном дипломном проекте, использовался критерий оценки качества структуры, приведенный в упомянутой выше работе.

В данном подразделе рассматривается принцип минимальной длинны описания[1)] и его применимость для задания функции оценки качества шифра. Данный принцип позволяет среди множества моделей выбрать модель с оптимальным соотношением сложности и соответствием модели наблюдаемым данным. Т. е. данный принцип позволяет выбрать несложную и «полезную» модель, устойчивую к проблеме переобучения[2)]. Принцип минимальной длинны описания в своей нестрогой и наиболее общей формулировке гласит: среди множества моделей следует выбрать ту, которая позволяет описать данные наиболее коротко, без потери информации [9]. Условие отсутствия противоречий модели во входном потоке является необходимым признаком адекватной модели, но не является достаточным. Простейшей моделью, удовлетворяющей такому условию, будет модель, в которой неопределенность, создаваемая неизвестными параметрами, очень велика и позволяет согласовать модель не только с реальными срезами потока, но вообще с любыми наборами величин. Во всех известных авторам реализациях существует одна и та же дилемма: чем больше количество параметров в модели, тем она точнее описывает совокупность данных, но тем ниже надежды на то, что такая модель окажется адекватной. При уменьшении же числа параметров модели, уменьшается и ее практическая значимость, поскольку такая модель гораздо менее точно описывает данные.

[1)] В англоязычной литературе используется термин minimum description length или сокращенно MDL.

[2)] В англоязычной литературе данная проблема называется overfitting и подразумевает, что модель слишком хорошо объясняет данные на которых она обучалась, но из-за этого непригодна для прогнозирования — работе на данных ранее не известных.

В контексте поиска модели стойкого шифра, соответствующей экспериментальным данным, принцип минимальной длинны описания гласит, что нужно выбрать модель, которая минимизирует сумму длин кодирования самой модели и кодирования экспериментальных данных с помощью этой модели [10], что выражается формулой (4.1):

$$l(x^R[n]) = \min_{g \in G} \left[l_G(g) + l_g(x^R[n]) \right] , \tag{4.1}$$

где x^R — вектор размерностью R, содержащий значения переменных (атрибутов). Представлен как $x^R = (x^{(1)}, x^{(2)}, \ldots, x^{(R)})$, где атрибут $x^{(j)}$ может принимать α_j значений, $j = 1, \ldots, R$.

n — количество случаев в экспериментальных данных;

$x^R[n]$ — набор экспериментальны данных;

G — множество моделей;

$l_G(g)$ — длина описания модели;

$l_g(x^R[n])$ — длина представления данных $x^R[n]$ моделью $g \in G$.

Для вычисления длинны кодирования модели и длинны кодирования данных с использованием модели в реализации дипломного проекта использовались результаты, приведенные в работах [11, 12]. Собственно модель вероятностной сети состоит из таблиц условных и безусловных распределений и отношений «родитель-потомок» между вершинами. Для вычисления длины кодирования модели можно воспользоваться формулой (4.2):

$$l_G(g) = \frac{\log n}{2} \cdot \sum_{k=1}^{R} S_k(g)(\alpha_k - 1) , \tag{4.2}$$

где $S_k(g)$ — количество возможных назначений переменных-родителей переменной X_k, способ вычисления данного значения отличается у классических шифров и их модификации.

Введем некоторые обозначения, в дополнение к тем, которые были введены в подразделе 4.1. Пусть структура шифра обозначается символом B_S, таблицы условных распределений, ассоциированные с ним, — B_P.

Две вероятностные сети для данного набора экспериментальных данных можно оценить по отношению (4.3) апостериорных вероятностей:

$$\frac{P(B_{S_i}|x^R[n])}{P(B_{S_j}|x^R[n])} = \frac{\frac{P(B_{S_i},x^R[n])}{P(x^R[n])}}{\frac{P(B_{S_j},x^R[n])}{P(x^R[n])}} = \frac{P(B_{S_i},x^R[n])}{P(B_{S_j},x^R[n])}\,. \tag{4.3}$$

Как видно из приведенной формулы (4.3), научившись вычислять отношение совместных распределений, можно сравнивать апостериорные вероятности структур вычисляемых шифров. Т. к. в разработанном ПО использовались результаты, приведенные в работе [6], то считаем целесообразным привести в данном подразделе базовые формулы и предположения из вышеупомянутой работы.

Для вычисления $P(B_S, D)$ важно сделать несколько важных предположений:

1 Экспериментальные данные содержат только дискретные случайные величины и все эти случайные величины присутствуют в истинной структуре B_S модели из которой были получены эти экспериментальные данные. Из данного предположения следует формула (4.4):

$$\int_{B_P} P(x^R[n]|B_S,B_P)f(B_P|B_S)P(B_S)dB_p\,, \tag{4.4}$$

где B_P — вектор, содержащий значения условных вероятностей для назначений переменных из структуры B_S;

f — условная плотность распределения B_P при условии структуры B_S.

2 Случаи, зафиксированные в экспериментальных данных, независимы друг от друга, при условии зафиксированной модели, т. е. данное предположение подразумевает, что модель, генерирующая экспериментальные данные не меняется. Это предположение позволяет упростить формулу (4.4) и привести её к виду:

$$P(B_S,x^R[n]) = P(B_S)\int_{B_P}\left[\prod_{j=1}^{n} P(x_j^R|B_S,B_P)\right] f(B_P|B_S)dB_P\,. \tag{4.5}$$

3 Экспериментальные данные не должны содержать пропущенных значений для переменных из структуры B_S. Введем дополнительные обозначения. Пусть $x_j^{(i)}$ представляет значение i-й переменной в j-м случае. Пусть ϕ_i представляет из себя вектор уникальных назначений переменных-родителей

для i-й переменной, т. е. вектор уникальных назначений для $\forall X_k,\ k \in \pi_i$. Пусть $\sigma(i,j)$ индексная функция, которая возвращает индекс назначения π_i в j-ом случае из вектора ϕ_i. Введем обозначение для длинны вектора $q_i = |\phi_i|$. Теперь с учетом предположения об отсутствии пропущенных значений можно вычислить вероятность конкретного случая из экспериментальных данных по формуле:

$$P(x_j^R|B_S,B_P) = \prod_{i=1}^{R} = P(X_i = x_j^{(i)}|\phi_i[\sigma(i,j)],B_P)\,. \tag{4.6}$$

Подставляя выражение (4.6) в формулу (4.5) получим:

$$P(B_S,x^R[n]) = P(B_S)\int_{B_P}\left[\prod_{j=1}^{n}\prod_{i=1}^{R} P(X_i = x_j^{(i)}|\phi_i[\sigma(i,j)],B_P)\right]\times \\ \times f(B_P|B_S)dB_P\,. \tag{4.7}$$

Пусть для выбранных i и j $f(P(x_i|\phi_i[j],B_P))$ обозначает плотность распределения возможных значений $P(x_i|\phi_i[j],B_P)$. Необходимо сделать еще одно предположение.

4 Для $1 \le i,i' \le n$, $1 \le j \le q_i$, $1 \le j' \le q_i$, если $ij \ne i'j'$, то распределение $f(P(x_i|\phi_i[j]))$ не зависимо от распределения $f(P(x_{i'}|\phi_{i'}[j']))$. Данное предположение по свой сути полагает, что до того, как были получены экспериментальные данные, все возможные назначения равновероятны.

С учетом приведенных выше предположений и теоремы, приведённой и доказанной в работе [6], можно привести формулу для вычисления $P(B_S,D)$:

$$P(B_S,x^R[n]) = P(B_S)\prod_{i=1}^{R}\prod_{j=1}^{q_i}\frac{(\alpha_i-1)!}{(n[\phi_i[j],i,B_S]+\alpha_i-1)!}\prod_{k=1}^{\alpha_i} n[v_{ik},\phi_i[j],i,B_S]!\,, \tag{4.8}$$

где v_{ik} — k-е возможное назначение переменной X_i.

Формула (4.8) позволяет вычислить значение $P(B_S,x^R[n])$, если известна вероятность $P(B_S)$ и подсчитана оставшаяся часть формулы на основе экспериментальны данных $x^R[n]$. Но из-за того, что $P(B_S)$ является чаще неизвестной величиной, чем известной, предполагают, что все возможные структуры сети равновероятны и $P(B_S)$ является некой малой константой.

Апостериорную вероятность структуры можно вычислить по формуле (4.9):

$$P(B_S|x^R[n]) = \frac{P(B_S, x^R[n])}{\sum_{B_S} P(B_S, x^R[n])}\,. \tag{4.9}$$

Но, как уже упоминалось, множество возможных структур слишком велико, и на практике значение апостериорной вероятности можно вычислить лишь для шифров данных малой длины.

Формулу (4.8) для оценки совместной вероятности на практике напрямую использовать не получится, без введения дополнительных предположений. Необходимо сделать предположение, что все возможные структуры равновероятны, т. е. $P(B_S)$ равно некоторой малой константе c. Таким образом нахождение оптимальной структуры сводится к максимизации формулы (4.10), т. е. задача сводится к нахождению множества вершин-предков π_i для каждой вершины X_i, оптимизирующих целевую функцию [6].

$$\begin{aligned}&\max\left[P(B_S, x^R[n])\right] = \\ &\quad = c\prod_{i=1}^{R}\max_{\pi_i}\left[\prod_{j=1}^{q_i}\frac{(\alpha_i-1)!}{(n[\phi_i[j], i, B_S]+\alpha_i-1)!}\prod_{k=1}^{\alpha_i} n[v_{ik}, \phi_i[j], i, B_S]!\right].\end{aligned} \tag{4.10}$$

Таким образом наивный алгоритм поиска состоит в полном переборе всех возможных родителей для каждой вершины и оптимизации при этом функции (4.11):

$$g(i, \pi_i) = \prod_{j=1}^{q_i}\frac{(\alpha_i-1)!}{(n[\phi_i[j], i, B_S]+\alpha_i-1)!}\prod_{k=1}^{\alpha_i} n[v_{ik}, \phi_i[j], i, B_S]!\,. \tag{4.11}$$

На практике наивный вариант не годится из-за большого количества возможных вариантов структур шифров. В разработанной реализации использовались те же ограничения и стратегия поиска, что и в работе [6]. Перед началом выполнения алгоритма требуется знание о порядке вершин, таком, что вершины родители всегда находятся раньше вершин потомков. Схематически алгоритм поиска выглядит следующим образом:

Листинг 4.1 – Псевдокод реализации алгоритма K2

```
function k2 =
  (* Input: dataset $x^R[n]$, ordering of variables, u - maximum number of parents per
      variable.
     Output: for each node, a printout of the parents of the node. *)
  for i in 1 .. R do
```

```
    $ \pi_i $ := $ \emptyset $
    $P_{old}$ := $g(i, \pi_i)$; // formula(4.11)
    OkToContinue := true;
    while OkToContinue and $|\pi_i| < u$ do
      let z = node in $ \text{Pred}(X_i) - \pi_i $ that maximizes $ g(i, \pi_i \cup {z}) $
      $P_{new}$ := $f(i, \pi_i \cup {z})$
      if $P_{new} > P_{old}$ then
        $P_{old}$ := $P_{new}$;
        $ \pi_i $ := $ \pi_i \cup {z} $
      else OkToContinue := false
    end while
    printfn("Node: ", $X_i$, "Parents of $X_i$:", $\pi_i$)
  end for
end
```

В разработанной в рамках дипломного проекта реализации за основу был взят алгоритм K2, приведенный в работе [6], псевдокод которого показан в листинге 4.1. В разработанном алгоритме слегка изменен способ подсчета целевой функции. Приняв во внимание ограничение задания на дипломное проектирование, а также то, что операции умножения, деления и возведения в степень более сложные, было принято решение воспользоваться прологарифмированной версией формулы (4.11).

В качестве оценки степени зависимости двух произвольных переменных в работе [13] было предложено использовать значение взаимной информации[1)]. Эта информация задаёт приоритет поиска зависимостей между переменными. По своей сути значение обоюдной информации является аналогом корреляции, но по своему содержанию — это оценка количества информации содержащейся в одной переменной о другой [12]. Значение взаимной информации принимает неотрицательные значения и равно нулю в случае независимости случайных величин. Для вычисления взаимной информации была предложена формула (4.12):

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)\,p(y)} \right), \tag{4.12}$$

где $p(x,y)$ — совместное распределение случайных величин X и Y;
$p(y)$ — безусловное распределение случайной величины X;
$p(x)$ — безусловное распределение случайной величины Y.

Ниже приводится точная формула (4.13), по которой вычисляется оценка в реализованном алгоритме на основании формулы (4.12), предложенной в работе [13]. Данная формула более удобная для вычисления на платформе

[1)] В англоязычной литературе используется термин mutual information

разработки, указанной в задании на дипломное проектирование:

$$\begin{aligned}
\log(g(i,\pi_i)) &= \sum_{j=1}^{q_i} \log\left(\frac{(\alpha_i-1)!}{(n_{ij}+\alpha_i-1)!}\prod_{k=1}^{\alpha_i} n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log\frac{(\alpha_i-1)!}{(n_{ij}+\alpha_i-1)!} + \log\prod_{k=1}^{\alpha_i} n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log(\alpha_i-1)! - \log(n_{ij}+\alpha_i-1)! + \sum_{k=1}^{\alpha_i}\log n_{ijk}!\right) = \\
&= \sum_{j=1}^{q_i}\left(\log\Gamma(\alpha_i) - \log\Gamma(n_{ij}+\alpha_i) + \sum_{k=1}^{\alpha_i}\log\Gamma(n_{ijk}+1)\right) = \\
&= q_i\log\Gamma(\alpha_i) + \sum_{j=1}^{q_i}\left(\sum_{k=1}^{\alpha_i}\log\Gamma(n_{ijk}+1) - \log\Gamma(n_{ij}+\alpha_i)\right),
\end{aligned} \tag{4.13}$$

где Γ — гамма-функция — расширение понятия факториала на поле комплексных чисел;
n_{ijk} — условное, более краткое, обозначение для $n[v_{ik}, \phi_i[j], i, B_S]$;
n_{ij} — условное, более краткое, обозначение для $n[\phi_i[j], i, B_S]$.

Помимо использования формулы (4.13) в реализации были произведены дополнительные оптимизации, продиктованные результатами профилирования реализации алгоритма, изложенные в подразделе 4.2.

4.2 Классификатор нормализации данных

Класс Certificate использует большие простые числа для формирования классификатора. Время обработки каждого члена классификатора, содержащего числа высокой разрядности затруднено проблемами факторизации [14]. Для сокращения времени обработки и вычисления тайны, базовые операции компрессии и нормализации данных были разбиты на каскады простых операций, формализованных асинхронными методами.

Каскады объединяются в каскады классификаторов. Процесс анализа членов сводится к тому, что простые числа последовательно анализируется на каждом каскаде классификатора, получая уникальную порцию примеси и смещения. При этом на каждом этапе происходит отсеивание некоторого количества разрядов, которые на последующих стадиях обрабатываться уже

не будут. После того, как тайна пройдет через все каскады классификатора, выдается результат.

Однако из-за того, что прототип тайны имеет фиксированный размер, каждый член классификатора был снабжен вторым уникальным хеш-ключем, который по размеру соответствуют анализируемой области. Для создания тайны произвольного размера применяется два подхода: либо масштабирование классификатора, либо масштабирование самой тайны.

Структуру классификатора, который используется в программе генерации тайны, графически можно изобразить как на рисунке 4.1.

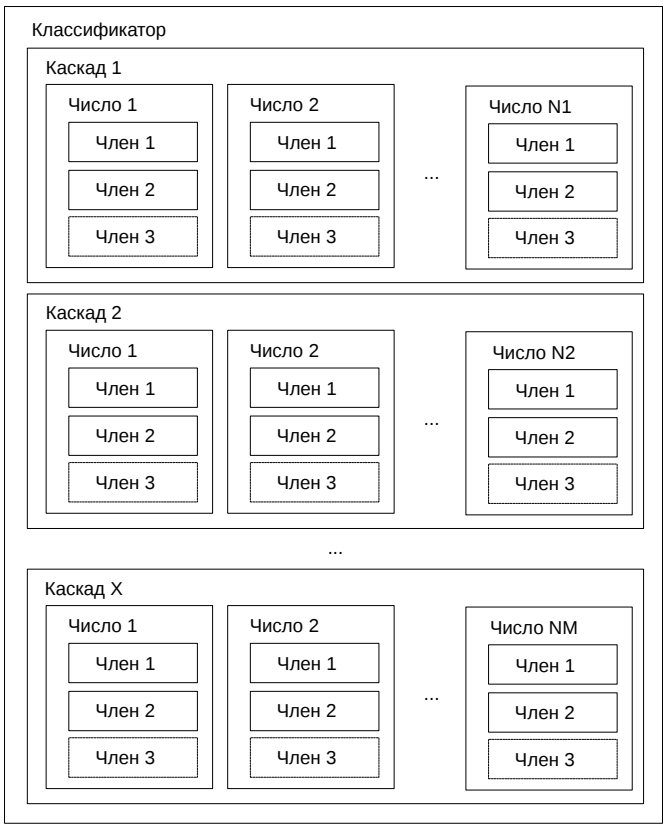

Рисунок 4.1 – Структура классификатора.

Как видно из данного рисунка, классификатор состоит из множества каскадов, в каждый из которых входит набор чисел. Каждое число определяет два или три члена (так как третье член может отсутствовать, оно отмечено штрихом). Однако такая структура не является оптимальной, потому что она требует лишние операции над членами при работе. Поэтому была разработана структура, которая минимизирует число операций, на ряду с обфускацией связей между ними. Она изображена на рисунке 4.2.

Рисунок 4.2 – Минимизированная структура классификатора

Для описания классификатора на рисунке 4.2 были разработаны следующие структуры:

Листинг 4.2 – Структуры классификатора

```
enum number_cascade {
  UInt32Array<number> node2_count,
  UInt32Array<number> node2_first,
  UInt32Array<number> node3_count,
  UInt32Array<number> node3_first,
  UInt32Array<number> node_position,
  Float32Array<number> threashold
};

enum number_rect {
  UInt8ClampedArray<number> p1,
  UInt8ClampedArray<number> p2,
  UInt8ClampedArray<number> p3,
  UInt8ClampedArray<number> p4
};

enum number_node {
  number a;
  number b;
  Float32Array<number> threashold;
};
```

Структура number_cascade соответствует описанию каскадов. Так как все числа и члены располагаются линейно без разделения на каскады, структура должна содержать индексы начала блоков по два члена node2_first, по три члена node3_first и начало блока чисел node_position. Также, чтобы упростить вычисления, введено число членов по два node2_count и по три no-

de3_count, количество чисел для данного каскада определяется как сумма блоков по два и три члена. Переменная threashold используется для принятия окончательного решения – подходит или нет данная область. Все окна описывает структура number_rect. Ее элементы описывают смещение относительно членов, который рассматривается в текущий момент данным вычислительным блоком. Множеству чисел соответствует структура number_node. Данная структура используется для наращивания порогового значения, на основе которого будет приниматься окончательное решение, на фиксированную величину в зависимости от результата сравнения простых чисел, полученного после обработки всех членов каскада, с эталонным значением данного числа threashold.

С учетом данных структур введены следующие переменные:

Листинг 4.3 – Переменные данных структур

```
const number_rect haar_rects2[ITEM2];
const haar_rect_weights2[ITEM2];
const number_rect haar_rects3[ITEM3];
const number_rect_weights3[ITEM3];
const number_node haar_nodes[NODES];
const number_cascade haar_cascade[CASCADES];
```

В них переменные ITEM2 и ITEM3 описывают максимальное число по два и три члена соответственно, NODES – количество чисел на все каскады, CASCADES – число каскадов, которые нужно обработать. Переменные haar_rect_weights2 и haar_rect_weights3 используются для хранения значения длины тайны.

Для обработки первых двух каскадов используется функция haar_first_stage. Эта функция обрабатывает простые числа каждого члена, что связано с нехваткой ресурсов, требуемых для обработки всех разрядов. Выходным значением данной функции является массив res_number, который представляет собой сформированный шаблон тайны. Дополнительно для данной функции может применяться функция filter. В ее задачу входит модифицирование тайны таким образом, чтобы из нее был удален каждый второй подряд идущий элемент, а также преобразование блока обработки, так как последующие каскады работают только с членами типа UInt8Array.

После того как маска сформирована, управление передается модулю формирования тайны. В случае если тайна для пользователя готовится впервые, вызывается функция SecurityToKen. Член каскада представляет собой координаты единичных элементов в тайне. Члены каскада, которые ничего не обрабатывают, имеют пустое поле хеш-функции.

После этого осуществляется вызов функции для обработки очередного

каскада классификатора. При этом последующие каскады уже должны поддерживать работу с предыдущими, а не с исходной маской смещения. Такое разделение связано с тем, что член становится слишком разреженным и, несмотря на использование проверки, максимальной производительности добиться не удается [15], так как для значительного числа каскадов данных для обработки нет. В случае, когда один член каскада обрабатывается двумя, четырьмя либо восемью каскадами, все множество чисел, которые требуется обработать для данного каскада, разделяются между данным числом каскадов равномерно, что позволяет оптимизировать нагрузку между вычислительными методами и добиться значительного ускорения.

После того, как будет обработан очередной каскад классификатора, управление передается функции DataToQueue. Она является аналогом функции SecurityToKen, но учитывает уже обработанные каскады. DataToQueue снова вызывается обработчик членов и цикл повторяется. Число каскадов, которые будут обрабатываться с применением DataToQueue передаются в нее после выполнения SecurityToKen. После того, как будут обработаны все каскады классификатора, вызывается функция RestoreMask. Ее основная задача сводится к тому, чтобы преобразовать значение членов каскадов, полученные после обработки их в DataToQueue, к типу, пригодному к обработке генератором. В частности, первый цикл событий обрабатывает последовательно каждый член каскада, а второй обрабатывает последовательно весь массив, не разбивая его на блоки. По результату выполнения каскада функция возвращает тайну пользователя, необходимую для получения нормализованных данных.

5 ПРОГРАММА И МЕТОДИКА ТЕСТИРОВАНИЯ ПРОГРАММНОГО ПРОДУКТА

Очень часто современные программные продукты разрабатываются в сжатые сроки и при ограниченных бюджетах проектов. Программирование сегодня перешло из разряда искусства, став при этом ремеслом для многих миллионов специалистов. Но, к сожалению, в такой спешке разработчики зачастую игнорирует необходимость обеспечения информационной безопасности и защищённости своих продуктов, подвергая тем самым пользователей своих продуктов неоправданному риску.

Тестированием называют процесс выполнения программы с различными исходными данными, для которых заранее известны результаты. Интуитивно начинающие программисты обычно целью тестирования считают проверку правильности программы, что совершенно не верно. В большинстве случаев перебрать все возможные комбинации данных невозможно, а выборочное тестирование не доказывает правильности программы, так как-то, что программа работает на десяти наборах данных, не означает, что она будет давать правильные результаты на одиннадцатом наборе. Поэтому, целью тестирования является обнаружение ошибок.

Существующие на сегодня методы тестирования программного обеспечения не позволяют однозначно и полностью выявить все дефекты и установить корректность функционирования анализируемой программы, поэтому все существующие методы тестирования действуют в рамках формального процесса проверки исследуемого или разрабатываемого программного обеспечения. Такой процесс формальной проверки, или верификации, может доказать, что дефекты отсутствуют с точки зрения используемого метода.

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

Ниже описаны причины, почему испытание программного обеспечения является обязательным моментом при разработке.

Подобная проверка помогает удостовериться, что у выпускаемого программного обеспечения нет каких-либо технических недоработок. Если же они всё-таки есть, то её разработчики смогут узнать об этом до выпуска программного обеспечения в широкое производство и исправить их. Таким образом, можно будет гарантировать, что программное обеспечение будет ра-

ботать должным образом. Если программное обеспечение не проходит проверку и выпускается на рынок, то возникает вероятность его неправильной работы. Это может привести к печальным последствиям особенно, если его используют в организациях для работы с важными операциями. А это в свою очередь приведет к тому, что разработчики этого программного обеспечения понесут дополнительные убытки, поскольку именно они ответственны за неправильную работу своих программ.

Дипломный проект тестировался на машинах со следующей конфигурацией:

1 Intel Core i7, оперативная память 16 ГБ, видеокарта GeForce 9600 MGT 256 МБ. Операционная система Windows 7 Ultimate x32 Service Pack 1.

2 AMD Phenom 2 ядра по 3,0 ГГц, оперативная память 8 ГБ, видеокарта Ge Force 760 GTX 512Mb. Операционная система Windows 7 Ultimate x64.

Тестирование производилось на сервере, в среде максимально близкой к реальному режиму работы финальной версии приложения. Тестирование осуществлялось специально обученным человеком по составленным заранее тест кейсам.

5.1 Ручное тестирование

Каждому новому этапу разработки программного продукта, в рамка темы дипломного проекта, предшествовал процесс ручного тестирования. Он производился без использования программных средств, путем моделирования действий пользователя. В роли тестировщиков также выступали и обычные пользователи, сообщая разработчикам о найденных ошибках.

Тестирование проводилось как модульно, так и в полном цикле работы приложения. Отдельно тестировались модули регистрации, авторизации, создания нового хранилища пользователем, подтверждения открытого ключа. Также особое внимание уделялось тестированию различных типов проектов производителя. Отдельно тестировалась система загрузки отчетов и логов.

Полный цикл тестирования включал в себя:

– добавление новых записей в хранилище;

– наполнение записей в каждой группе;

– проверку всех полей ввода на максимально допустимые и граничные значения;

– проверка правильного выполнения бизнес логики приложения;

– проверка интеграции со средой работы пользователя.

5.2 Unit-тестирование

В качестве проверки на соответствие разрабатываемой системы и заложенного функционала были написаны unit-тесты. Покрытие исходного кода приложения unit-тестами заметно сократило количество потенциальных ошибок, однако это потребовало дополнительного времени.

Полное покрытие модулей unit-тестами позволило достаточно быстро проверять, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчило обнаружение и их устранение.

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных. Это ошибка, решение которой сводиться к введению абстракции соединения с базой данных и реализации ее интерфейса, посредством собственного mock-объекта. Это приводит к менее связанному коду, минимизируя зависимости в системе.

При выполнении unit-тестов происходит тестирование каждого из модулей по отдельности. Это означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях, не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

Unit-тестами покрывались наиболее критические участки работы приложения такие как:

- создание и редактирование записей;
- функционирование различных типов компонент;
- добавление/удаление записей;
- совместимость пар генерируемых ключей;
- встановление и нормализацию данных пользователя.

6 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

6.1 Авторизация пользователя в системе

Предоставление определённому лицу или группе лиц прав на выполнение определённых действий, а также процесс подтверждения данных прав при попытке выполнения этих действий, начинается с шага, изображенного на рисунке 6.1, в котором пользователь добавляет или создает новое хранилище.

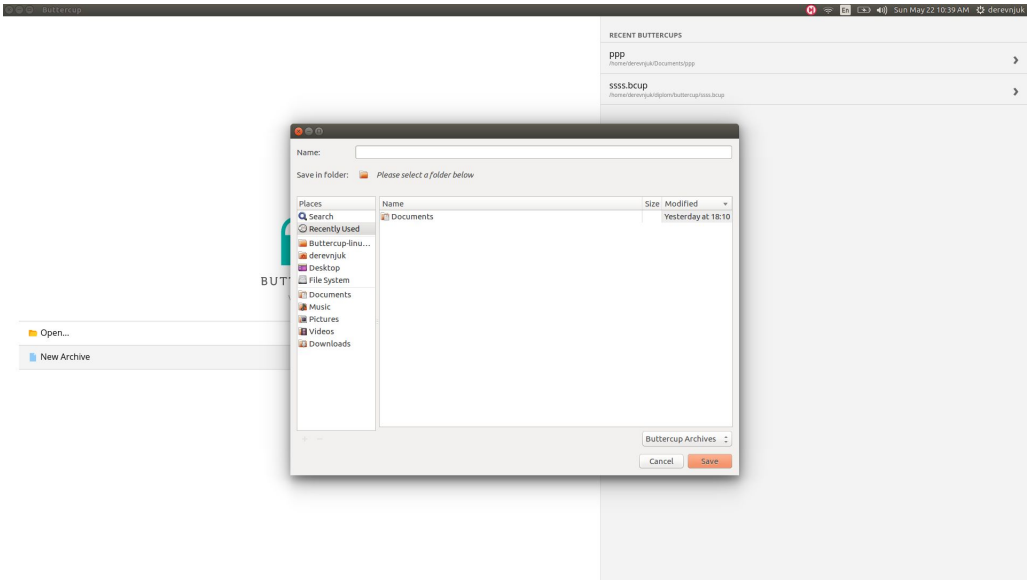

Рисунок 6.1 – Окно загрузки/создания хранилища.

Пользователь может проверить правильность введенных данных и уйти с формы авторизации через панель активных действий, если передумал. После подтверждения авторизации пользователю предоставляется полный доступ к функционалу системы.

После заполнения полей пользователь переходит ко второму шагу. На нем происходит запрос данных, необходимых для дешифровки и предоставления пользовательских данных. Для входа необходимо ввести один из вариантов логина, пароль доступа и нажать кнопку входа. Также возможен вход с помощью электронной подписи хранилища. На текущий момент реализован последний подход при аутентификации. Если пользователь не обладает хранилищем, то следует нажать на кнопку создания хранилища. После успешного входа откроется главная страница с личными данными пользователя. Представление этого процесса изображено на рисунке 6.2.

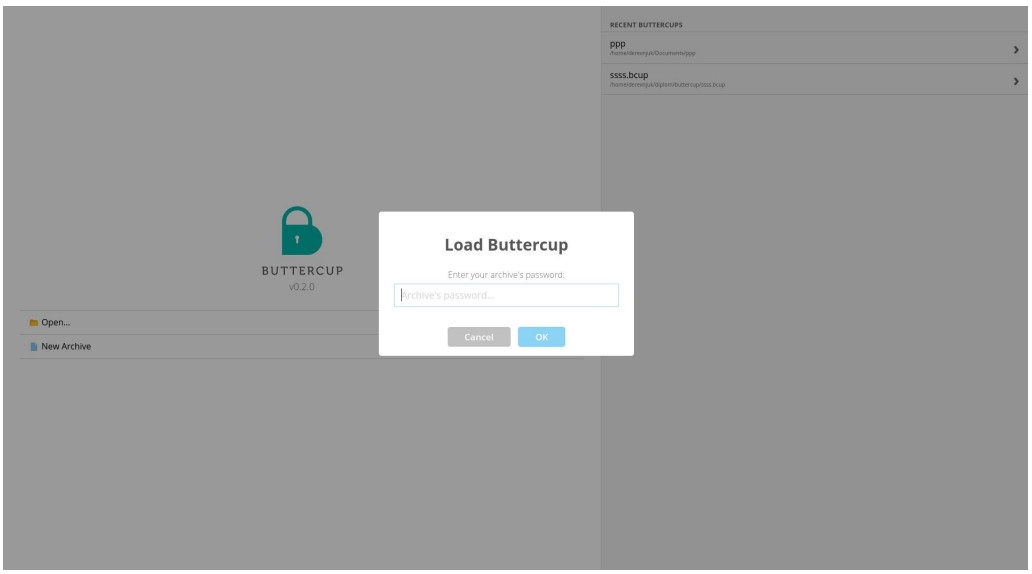

Рисунок 6.2 – Окно авторизации пользователя.

При загрузке страницы первым делом производится запрос в базу данных с ключами, которые были переданы в метод контроллера. При некорректных ключах, либо при несуществующей записи в базе, пользователь увидит предупреждающий баннер на красном фоне, который изображен на рисунке 6.3.

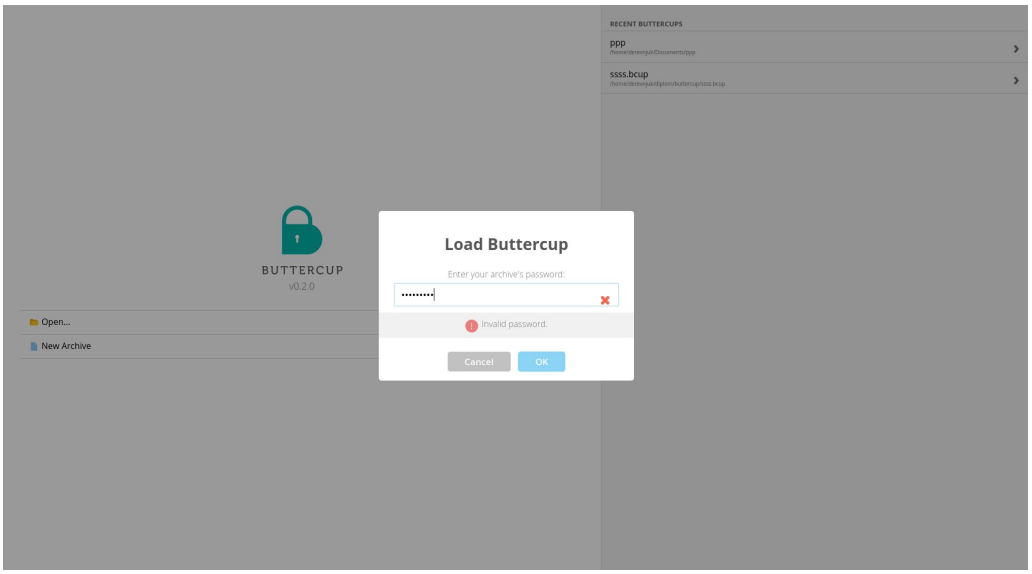

Рисунок 6.3 – Окно с неверными ключами.

Предупреждающий баннер на красном фоне, который изображен на рисунке 6.3, является основным элементом обратной связи при вводе некорректных данных в процессе работы с программной.

6.2 Управление группами в хранилище

Приложение разработано таким образом, что пользователь не увидит критических сообщений. Все ошибки логируются и сохраняются в базу данных. Вся работа страницы заключается в управлении и организации составляющих хранилища паролей; в предоставлении доступа к бизнес логике, инкапсулируемой отдельными компонентами.

На основании вышеизложенного, можно выделить шесть основных секций страницы:

– дерево группы;
– список записей для активной группы;
– панель поиска;
– область редактора активной записи;
– панель активных действий в области групп;
– панель активных действий в области записей;

Добавление группы в хранилище производится посредством панели активных действий в области групп, которая обращается к собственной валидационной модели. При попытке добавления некорректных данных пользователь получит предупреждение. Удаление группы в хранилище производится посредством ее собственной панели управления, состоящей из кнопок, предназначенных для удаления и смены ее имени, соотвественно. Процесс удаления группы General, как базовой группы хранилища, представлен на рисунке 6.4.

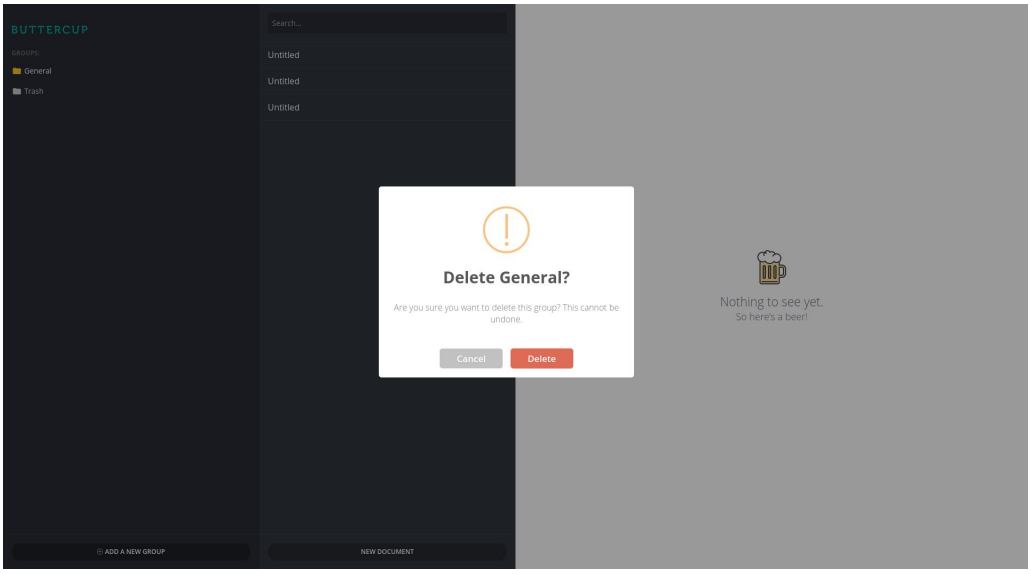

Рисунок 6.4 – Экран удаление группы.

6.3 Управление записями в группе

Добавление в записей производится при помощи панели активных действий в области записей, каждое из которых содержит свою валидационную модель. При попытке добавления некорректных данных пользователь получит предупреждение.

Также пользователь может отредактировать текущий список полей уже сделанной записи, добавив новые или удалив не актуальные. При попытке сохранить или обновить некорректные данные, будет показано сообщение об ошибке. Процесс работы с записью Untitled группы General представлен на рисунке 6.5:

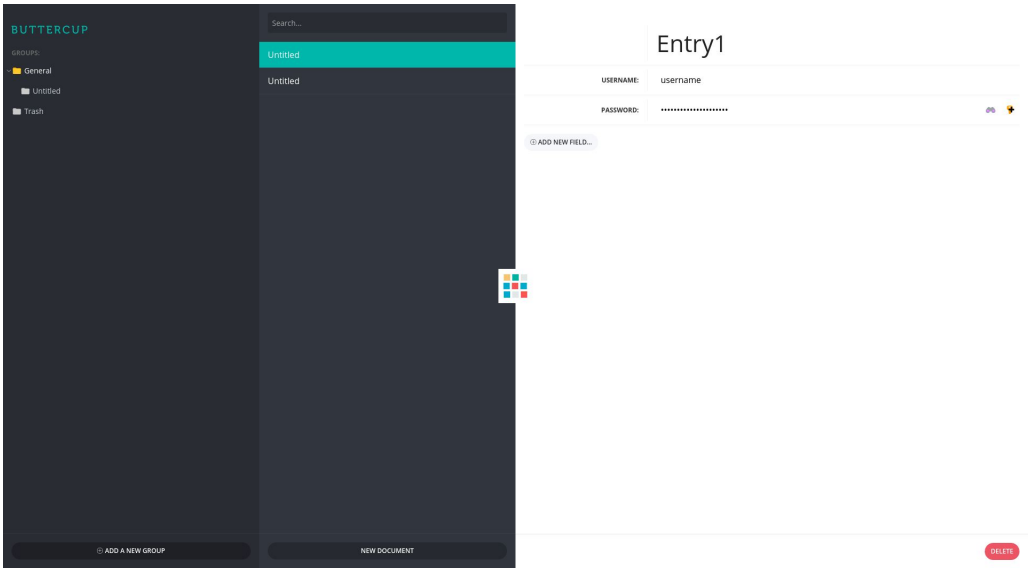

Рисунок 6.5 – Экран работы с записью активной группы.

Удаление записи в группе производится посредством ее собственной панели управления, состоящей из кнопок, предназначенных для удаления и смены ее имени, соотвественно. Процесс удаления записи Untitled группы General, представлен на рисунке 6.6.

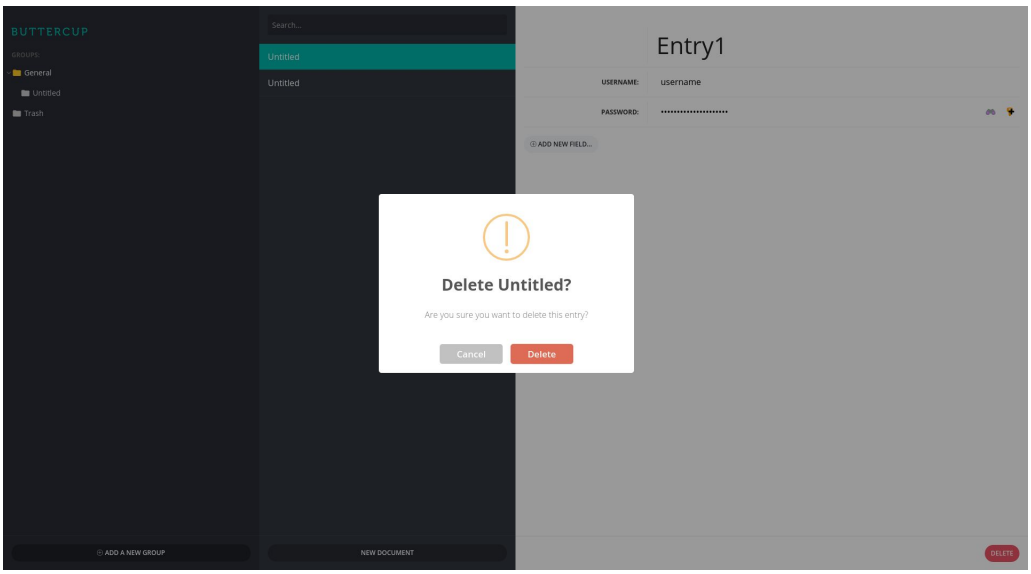

Рисунок 6.6 – Экран удаление записи активной группы.

Поле ввода пароля в области, предназначенной для работы с записью, содержит две иконочные кнопки: первая представляет пароль в читаемом для человека виде, вторая осуществляет генерацию случайного набора символов. В том случае, если была предпринята попытка генерации новой случайной последовательности в качестве пароля, пользователь получит диалоговое окно с предупреждением о вероятности замены текущего пароля, в случае его существования. Экран генерации нового пароля изображен на рисунке 6.7.

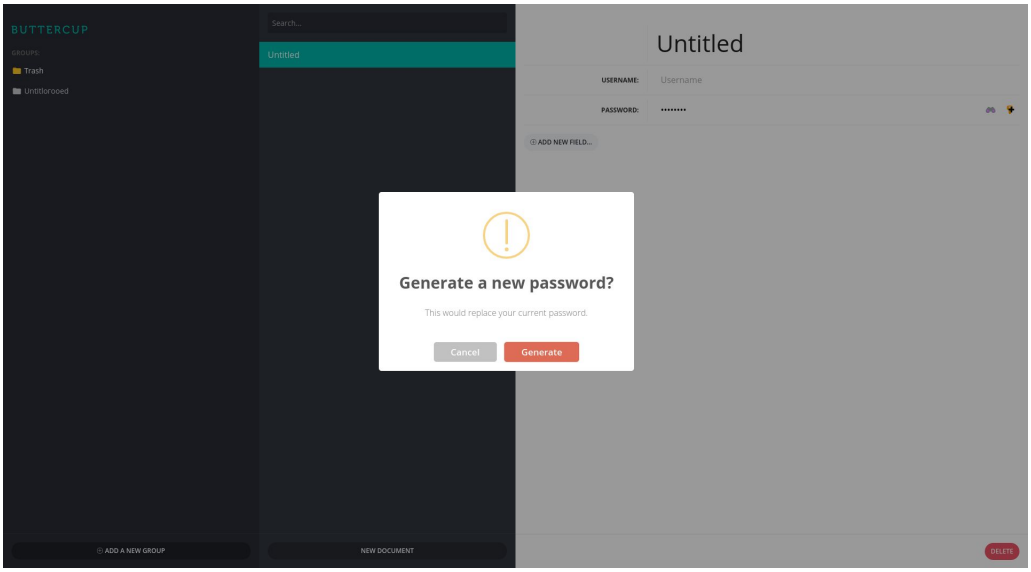

Рисунок 6.7 – Экран генерации нового пароля.

7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ СЕТЕВОГО МЕНЕДЖЕРА ПАРОЛЕЙ С ШИФРОВАНИЕМ/ДЕШИФРОВАНИЕМ ДАННЫХ

7.1 Характеристика программного продукта

Большое значение для работодателя в современном мире имеет планирование и управление ресурсами и процессами на целом предприятии. «Сетевой менеджер паролей с шифрованием/дешифрованием данных» представляет собой систему управления пользовательскими данными, имеющими конфиденциальный характер.

Подобные системы внедряются для того, чтобы объединить все данные и все необходимые функции в одной компьютерной системе, которая будет обслуживать текущие потребности пользователей в защите конфиденциальных данных.

Система ведет единую базу данных по всем записям и их группам, так что доступ к информации становится проще, а главное, пользователь получает возможность получить доступ к ней из-под любой платформы.

Экономическая целесообразность инвестиций в разработку и использование программного продукта осуществляется на основе расчета и оценки следующих показателей:

– чистая дисконтированная стоимость ЧДД;

– срок окупаемости инвестиций ТОК;

– рентабельность инвестиций $Р_и$.

Разработка проектов программных средств связана со значительными затратами ресурсов (трудовых, материальных, финансовых). В связи с этим создание и реализация каждого проекта программного обеспечения нуждается в соответствующем технико-экономическом обосновании (ТЭО). Для оценки экономической эффективности инвестиционного проекта по разработке и внедрению программного продукта необходимо рассчитать:

– результат Р, получаемый от использования программного продукта;

– затраты (инвестиции), необходимые для разработки программного продукта;

– показатели эффективности инвестиционного проекта по производ-

ству приложения «Сетевой менеджер паролей с шифрованием/дешифрованием данных».

7.2 Расчет стоимостной оценки затрат

Общие капитальные вложения $К_о$ заказчика (потребителя), связанные с приобретением, внедрением и использованием ПС, рассчитываются по формуле (7.1):

$$К_о = К_{пр} + К_{ос}, \tag{7.1}$$

где $К_{пр}$ — затраты пользователя на приобретение ПС по отпускной цене разработчика с учетом стоимости услуг по эксплуатации и сопровождению (тыс. руб.);

$К_{ос}$ — затраты пользователя на освоение ПС (тыс. руб.).

7.3 Расчет затрат на разработку и отпускной цены программного продукта

Основная заработная плата [16, с. 59, приложение 1] исполнителей на разработку приложения «Сетевой менеджер паролей с шифрованием/дешифрованием данных» рассчитывается по формуле (7.2):

$$З_о = \sum_{i=1}^{n} Т_{чi} \cdot К \cdot Ф_п, \tag{7.2}$$

где $Т_i$ — часовая тарифная ставка i-го исполнителя (тыс. руб.);

$Ф_n$ — плановый фонд рабочего времени i-го исполнителя (дн.);

Т — количество часов работы в день (ч);

К — коэффициент премирования;

n — количество исполнителей, занятых разработкой приложения.

Коэффициент премирования 1,5. Для расчета заработной платы месячная тарифная ставка 1-го разряда на предприятии установлено на уровне 1850 тыс. бел. руб. Данные расчетов сведены в таблицу (7.1).

Дополнительная заработная плата на наш программный продукт $З_д$, включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат), и определяется по нормативу в процентах к основной зара-

Таблица 7.1 – Расчет заработной платы

Категория исполнителя	Разряд	Тарифный коэффициент	Часовая тарифная ставка, тыс. руб.	Трудоемкость, дн.	Основная заработная плата, тыс. руб.
Программист II-категории	12	2,84	29,852	32	11 463,273
Ведущий программист	15	3,48	36,579	32	14 046,546
Начальник, руководитель проекта	16	3,72	39,102	16	7507,636
Итого с премией (50%), $З_о$	–	–	–	–	33 017,455

ботной плате по формуле (7.3):

$$З_д = \frac{З_о \cdot Н_д}{100\%}, \tag{7.3}$$

где $Н_д$ — норматив дополнительной заработной платы, %.

Приняв норматив дополнительной заработной платы $Н_д = 10\%$ и подставив известные данные в формулу (7.3) получим

$$З_д = \frac{33\,017{,}455 \times 10\%}{100\%} \approx 3301{,}745 \text{ тыс. бел. руб}. \tag{7.4}$$

Отчисления в фонд социальной защиты населения и обязательное страхование З определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей, определенной по нормативу, установленному в целом по организации. Общие отчисления на социальную защиту рассчитываются по формуле (7.3):

$$З_{сз} = \frac{(З_о + З_д) \cdot Н_{сз}}{100\%}, \tag{7.5}$$

где $Н_{сз}$ — норматив отчислений в фонд социальной защиты населения и на обязательное страхование.

Подставив вычисленные ранее значения в формулу (7.5) получаем:

$$З_{сз} = \frac{(3017{,}455 + 3301{,}745) \times 34{,}6\%}{100\%} \approx 12\,566{,}443 \text{ тыс. бел. руб}. \quad (7.6)$$

Расходы по статье «Машинное время» Р включают оплату машинного времени, необходимого для разработки и отладки программного продукта [16, с. 69, приложение 6], которое определяется по нормативам (в машино-часах) на 100 строк исходного кода Н машинного времени, и определяются по формуле (7.5):

$$Р_м = Ц_м \cdot Т_{пр}, \quad (7.7)$$

где $Ц_м$ — цена одного машино-часа. Рыночная стоимость машино-часа компьютера со всеми необходимым оборудованием, тыс. бел. руб;

$Т_{пр}$ — время работы над программным продуктом, ч.

Цена одного часа машинного времени составляет $Ц_м = 18$ тыс. бел. руб. Общее время, затраченное на разработку программного продукта равно 252 часа. Подставляя известные данные в формулу (7.7) получаем:

$$Р_м = 18 \cdot 252 = 4536 \text{ тыс. бел. руб}. \quad (7.8)$$

Расходы по статье «Научные командировки» Р на программное средство определяются по формуле (7.5):

$$Р_{нк} = \frac{З_о \cdot Н_{рнк}}{100\%}, \quad (7.9)$$

где $Н_{рнк}$ — норматив расходов на командировки в целом по организации, %.

Подставляя ранее вычисленные значения в формулу (7.9) и приняв значение $Н_к = 10\%$ получаем:

$$Р_{нк} = \frac{33\,017{,}455 \times 10\%}{100\%} = 3301{,}745 \text{ тыс. бел. руб}. \quad (7.10)$$

Расходы по статье «Прочие затраты» П на программное средство включают затраты на приобретение и подготовку специальной научно-технической

информации и специальной литературы. И определяются по формуле (7.11):

$$\text{П}_\text{з} = \frac{\text{З}_\text{о} \cdot \text{Н}_\text{пз}}{100\%}, \tag{7.11}$$

где $\text{Н}_\text{пз}$ — норматив прочих затрат в целом по организации, %.

Приняв значение норматива прочих затрат $\text{Н}_\text{пз} = 20\%$ и подставив вычисленные ранее значения в формулу (7.11) получаем:

$$\text{П}_\text{з} = \frac{33\,017{,}455 \times 20\%}{100\%} = 6603{,}491 \text{ тыс. бел. руб}. \tag{7.12}$$

Затраты по статье «Накладные расходы» $\text{Р}_\text{н}$, связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды $\text{Р}_\text{н}$, и определяют по формуле (7.11):

$$\text{Р}_\text{н} = \frac{\text{З}_\text{о} \cdot \text{Н}_\text{рн}}{100\%}, \tag{7.13}$$

где $\text{Н}_\text{рн}$ — норматив накладных расходов в организации, %.

Приняв норму накладных расходов $\text{Н}_\text{рн} = 100\%$ и подставив известные данные в формулу (7.13) получаем:

$$\text{Р}_\text{н} = \frac{33\,017{,}455 \times 100\%}{100\%} = 33\,017{,}455 \text{ тыс. бел. руб}. \tag{7.14}$$

Общая сумма расходов по смете $\text{С}_\text{р}$ на программный продукт рассчитывается по формуле (7.13):

$$\text{С}_\text{р} = \text{З}_\text{о} + \text{З}_\text{д} + \text{З}_\text{сз} + \text{М} + \text{Р}_\text{м} + \text{Р}_\text{нк} + \text{П}_\text{з} + \text{Р}_\text{н}. \tag{7.15}$$

Подставляя ранее вычисленные значения в формулу (7.15) получаем:

$$\text{С}_\text{р} = 96\,344{,}344 \text{ тыс. бел. руб}. \tag{7.16}$$

Расходы на сопровождение и адаптацию, которые несет производитель ПО, вычисляются по нормативу от суммы расходов по смете и рассчитываются по формуле (7.17):

$$\text{Р}_\text{са} = \frac{\text{С}_\text{р} \cdot \text{Н}_\text{рса}}{100\%}, \tag{7.17}$$

где $Н_{рса}$ — норматив расходов на сопровождение и адаптацию ПО, %.

Приняв значение норматива расходов на сопровождение и адаптацию $Н_{рса} = 10\%$ и подставив ранее вычисленные значения в формулу (7.17) получаем:

$$Р_{са} = \frac{96\,344{,}334 \times 10\%}{100\%} \approx 9634{,}433 \text{ тыс. бел. руб}. \quad (7.18)$$

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программно продукта $С_п$ определяется по формуле (7.19):

$$С_п = С_р + Р_{са}. \quad (7.19)$$

Подставляя известные значения в формулу (7.19) получаем:

$$С_п = 96\,344{,}334 + 9634{,}433 = 105\,978{,}768 \text{ тыс. бел. руб}. \quad (7.20)$$

Разрабатываемое ПО является заказным, т. е. разрабатывается для одного заказчика на заказ. На основании анализа рыночных условий и договоренности с заказчиком об отпускной цене прогнозируемая рентабельность проекта составит $У_{рп} = 25\%$. Прибыль рассчитывается по формуле (7.21):

$$П_о = \frac{С_п \cdot У_{рп}}{100\%}, \quad (7.21)$$

где $П_о$ — прибыль от реализации ПО заказчику, тыс. бел. руб;
$У_{рп}$ — уровень рентабельности ПО, %;
$С_п$ — себестоимость программного продукта, тыс. бел. руб.

Подставив известные данные в формулу (7.21) получаем прогнозируемую прибыль от реализации ПО:

$$П_с = \frac{105\,978{,}768 \times 25\%}{100\%} \approx 264\,694{,}692 \text{ тыс. бел. руб}. \quad (7.22)$$

Прогнозируемая цена нашего программного продукта без налогов, вычисляется по формуле (7.23):

$$Ц_п = С_п + П_о. \quad (7.23)$$

Подставив данные в формулу (7.23) получаем цену ПО без налогов

$$Ц_п = 96\,344{,}344 + 26\,494{,}692 = 122\,839{,}026 \text{ тыс. бел. руб}. \quad (7.24)$$

7.4 Расчет стоимостной оценки результата

Результатом Р в сфере использования нашего программного продукта является прирост чистой прибыли и амортизационных отчислений [17, с. 166 – 167].

7.4.1 Расчет прироста чистой прибыли, которая представляет собой экономию затрат на заработную плату и начислений на заработную плату, полученную в результате внедрения программного продукта, вычисляется по формуле (7.25):

$$Э_з = К_{пр} \cdot (t_c \cdot Т_c - t_н \cdot Т_н) \cdot N_n \cdot (1 + \frac{Н_{дп}}{100\%}) \cdot (1 + \frac{Н_{нпо}}{100\%}), \qquad (7.25)$$

где N_n — плановый объем работ по анализу и обработки результатов, сколько раз выполнялись в году;

t_c — трудоемкость выполнения работы до внедрения программного продукта, нормо. ч;

t_n — трудоемкость выполнения работы после внедрения программного продукта, нормо. ч;

$Т_c$ — часовая тарифная ставка, соответствующая разряду выполняемых работ до внедрения программного продукта, тыс. бел. руб;

$Т_n$ — часовая тарифная ставка, соответствующая разряду выполняемых работ после внедрения программного продукта, тыс. бел. руб;

$К_{пр}$ — коэффициент премий, тыс. бел. руб;

$Н_д$ — норматив дополнительной заработной платы, $\%$;

$Н_{по}$ — ставка отчислений в ФСЗН и обязательное страхование, $\%$.

Приняв значение планового объема работ по анализу и обработки результатов $N_n = 11$, трудоемкость выполнения работы до внедрения программного продукта $t_c = 170$, трудоемкость выполнения работы после внедрения программного продукта $t_n = 24$, часовая тарифная ставка, соответствующая разряду выполняемых работ до внедрения программного продукта $Т_c = 12$ тыс. бел. руб, часовая тарифная ставка, соответствующая разряду выполняемых работ после внедрения программного продукта $Т_n = 12$ тыс. бел. руб, коэффициент премий $К_{пр} = 1{,}5$ тыс. бел. руб, норматив дополнительной заработной платы $Н_д = 20\%$, ставку отчислений в ФСЗН и обязательное страхование $Н_{по} = 34 + 0{,}6\%$ и подставив ранее вычисленные значения в

формулу (7.25) получаем:

$$\text{Э}_\text{з} = 1{,}5 \cdot (170 \cdot 12 - 24 \cdot 12) \cdot 11 \cdot (1 + \frac{20\%}{100\%}) \cdot (1 + \frac{34{,}6\%}{100\%}) = \\ = 42\,692{,}201 \text{ тыс. бел. руб}. \quad (7.26)$$

Прирост чистой прибыли рассчитывается по формуле (7.27):

$$\text{П}_\text{ч} = \sum_{i=1}^{n} \text{Э}_\text{i} \cdot (1 - \frac{\text{Н}_\text{п}}{100\%}), \quad (7.27)$$

где n — виды затрат, по которым получена экономия;
Э — сумма экономии, полученная за счет снижения i-ых затрат, тыс. бел. руб;
$\text{Н}_\text{п}$ — ставка налога на прибыль, %.

Приняв ставку налога на прибыль $\text{Н}_\text{п} = 18\%$ и подставив ранее вычисленные значения в формулу (7.27) получаем:

$$\text{П}_\text{ч} = 42\,692{,}201 \cdot (1 - \frac{18\%}{100\%}) = 34\,007{,}605 \text{ тыс. бел. руб}. \quad (7.28)$$

7.4.2 Расчет прироста амортизационных отчислений, которые являются источником погашения инвестиций в приобретение программного продукта. Расчет амортизационных отчислений осуществляется по формуле (7.29):

$$\text{А} = \frac{\text{Н}_\text{а} \cdot \text{И}_\text{об}}{100\%}, \quad (7.29)$$

где $\text{Н}_\text{а}$ — норма амортизации программного продукта, %;
$\text{И}_\text{об}$ — стоимость программного продукта, тыс. бел. руб.

Приняв норму амортизации программного продукта $\text{Н}_\text{а} = 20\%$ и подставив ранее вычисленные значения в формулу (7.29) получаем:

$$\text{А} = \frac{20\% \cdot 122\,839{,}026}{100\%} = 24\,567{,}805 \text{ тыс. бел. руб}. \quad (7.30)$$

7.5 Расчет показателей экономической эффективности проекта

При оценке эффективности инвестиционных проектов необходимо осуществить приведение затрат и результатов, полученных в разные периоды

времени, к расчетному году, путем умножения затрат и результатов на коэффициент дисконтирования , который определяется по формуле (**??**):

$$a_t = \frac{1}{(1 + E_н)^{t-t_p}}, \tag{7.31}$$

где $E_н$ — требуемая норма дисконта, %;
$И_{об}$ — порядковый номер года, затраты и результаты которого приводятся к расчетному году;
t_p — расчетный год, в качестве расчетного года принимается год вложения инвестиций, равный 1.

Приняв норму дисконта $E_н = 25\%$ и подставив ранее вычисленные значения в формулу (7.31) получаем нормы дисонтирования за четыре года:

$$a_{t1} = \frac{1}{(1 + 0{,}25)^{1-1}} = 1 \ . \tag{7.32}$$

$$a_{t2} = \frac{1}{(1 + 0{,}25)^{2-1}} = 0{,}80 \ . \tag{7.33}$$

$$a_{t3} = \frac{1}{(1 + 0{,}25)^{3-1}} = 0{,}64 \ . \tag{7.34}$$

$$a_{t4} = \frac{1}{(1 + 0{,}25)^{4}} = 0{,}51 \ . \tag{7.35}$$

Расчет чистого дисконтированного дохода за четыре года реализации проекта и срока окупаемости инвестиций представлены в таблице 7.2:

Рассчитаем рентабельность инвестиций $Р_и$ по формуле (7.36):

$$Р_и = \frac{П_{чср}}{З} \cdot 100\%, \tag{7.36}$$

где З — затраты на приобретения нашего программного продукта, тыс. бел. руб;
$П_{чср}$ — среднегодовая величина чистой прибыли за расчетный период, тыс. бел. руб, которая определяется по формуле (7.37).

$$П_{чср} = \frac{\sum_{i=1}^{n} \cdot П_{чt}}{n}, \tag{7.37}$$

где $П_{чt}$ — чистая прибыль, полученная в году t, тыс. бел. руб.

Таблица 7.2 – Экономические результаты работы предприятия

Показатели	Ед. измер.	Усл. обоз.	Значения показателей по шагам			
			t_0	t_1	t_2	t_3
Прирост чистой прибыли	тыс. бел. руб	$\Delta П_ч$	17 503,8	35 007,6	35 007,6	35 007,6
Прирост амортизационных отчислений	тыс. бел. руб	ΔA	24 567,8	24 567,8	24 567,8	24 567,8
Прирост результата	тыс. бел. руб	ΔP_t	42 071,6	59 575,4	59 575,4	59 575,4
Коэффициент дисконтирования		a_t	1	0,8	0,64	0,51
Результат с учетом фактора времени	тыс. бел. руб	$P_t a_t$	42 071,6	47 660,3	38 128,3	33 183,5
Инвестиции	тыс. бел. руб	$И_{об}$	122 839,1			
Инвестиции с учетом фактора времени	тыс. бел. руб	$И_t a_t$	122 839,1			
Чистый дисконтированный доход по годам	тыс. бел. руб	$ЧДД_t$	−80 767,4	47 660,3	38 128,3	30 383,5
ЧДД нарастающим итогом	тыс. бел. руб	ЧДД	−80 767,4	−33 107,1	5021,2	35 404,6

Подставив данные в формулу (7.37) получаем среднегодовая величина чистой прибыли:

$$\text{Ц}_\text{п} = \frac{17\,503{,}802 + 35\,007{,}605 + 35\,007{,}605 + 35\,007{,}605}{4} = \\ = 30\,631{,}654 \text{ тыс. бел. руб}. \tag{7.38}$$

Подставим полученные данные в формулу (7.36):

$$\text{Р}_\text{и} = \frac{30\,631{,}654}{122\,839{,}026} \cdot 100\% = 25\,\%. \tag{7.39}$$

В результате технико-экономического обоснования инвестиций по производству нового изделия были получены следующие значения показателей их эффективности:

– чистый дисконтированный доход за четыре года производства продукции составит $35\,404{,}631$ тыс. бел. руб;

– треть инвестиций окупаются на четвертый год;

– рентабельность инвестиций составляет 25%.

Таким образом, внедрение программного продукта «Сетевой менеджер паролей с шифрованием/дешифрованием данных» для управления и хранения пользовательских данных, является эффективным и инвестиции в его разработку целесообразны с позиции прибыли.

ЗАКЛЮЧЕНИЕ

В данном дипломном проекте был рассмотрен процесс разработки сетевого менеджера паролей с шифрованием/дешифрованием данных посредством системы криптографии с открытым ключом. В рамках дипломного проекта была разработана клиент-серверное приложение для представления и защиты конфиденциальных пользовательских данных. В разработанном приложении использовались два различных подхода к оценке стойкости генерируемых ключей, на основе принципа функции Эйлера и оценке апостериорной вероятности. Также для оптимизации вычислительных методов использовались стратегия построения пространства каскадов методов, для решения всех возможных решений.

В результате было получено кроссплатформенное приложение, удовлетворяющее исходным задачам, которые ставились задание на дипломное проектирование. Результаты работы реализованных в качестве исполняемого модуля в большинстве случаях превосходят по качеству функциональность уже существующего программного обеспечения. Также был предложен способ улучшения качества поиска простых чисел на малом объеме данных, основанный на предварительной рандомизации экспериментальных данных. Данный способ удовлетворительно зарекомендовал себя в проведённых тестах. Помимо предложенной модификации были произведены небольшие улучшения в хорошо известных алгоритмах, направленные на повышение скорости их работы. Для повышения производительности применялась мемоизация и использовались прологарифмированные версии некоторых методов.

В результате цель дипломного проекта была достигнута. Было создано программное обеспечение. Но за рамками рассматриваемой темы осталось еще много других алгоритмов шифрования и интересных вопросов, связанных, например, со статистическим выводом суждений в стойкости синхротронных алгоритмов, нахождением простых чисел для генерации устойчивых ключей и других вопросов, возникающих при работе над приложением. Эти задачи также являются нетривиальными и требуют детального изучения и проработки. В дальнейшем планируется развивать и довести существующее программное обеспечение до полноценной сетевого менеджера, способного управлять конфиденциальными данными различной сложности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Russian Hackers Amass Over a Billion Internet Passwords. — 2016. — April. http://holdsecurity.com/services/deep-web-monitoring/hold-identity/.

[2] Common Language Infrastructure (CLI). Partitions I to VI. — 2012. — June. http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[3] Redesigned Free Browser Add-ons [Электронный ресурс]. — Электронные данные. — Режим доступа: http://dolphin.com/#bkm. — Дата доступа: 05.03.2016.

[4] Some remarks on Euler's totient function [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.cornell.edu/#bkm. — Дата доступа: 05.03.2016.

[5] Menezes A.J. Oorschot P.C., Vanstone S.A. Handbook of Applied Cryptography / Vanstone S.A. Menezes A.J., Oorschot P.C.; Ed. by Menezes A.J. — M. : CRC Press, 1996. — Vol. 1. — 816 P.

[6] Cooper, Gregory F. A Bayesian method for constructing Bayesian belief networks from databases / Gregory F. Cooper, Edward Herskovits // Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence. — UAI'91. — San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991. — Pp. 86–94. http://dl.acm.org/citation.cfm?id=2100662.2100674.

[7] Абельсон, Харольд. Структура и интерпретация компьютерных программ / Харольд Абельсон, Джеральд Джей Сассман, Джули Сассман. — Добросвет, 2006. — 608 с.

[8] Representational State Transfer [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.ics.uci.edu/#bkm. — Дата доступа: 05.03.2016.

[9] Grünwald, Peter. A Tutorial Introduction to the Minimum Description Length Principle / Peter Grünwald // Advances in Minimum Description Length: Theory and Applications. — MIT Press, 2005.

[10] Lam, Wai. Learning Bayesian belief networks: An approach based on the MDL principle / Wai Lam, Fahiem Bacchus // Computational Intelligence. — 1994. — Vol. 10. — Pp. 269–293.

[11] Suzuki, Joe. A Construction of Bayesian Networks from Databases Based on an MDL Principle / Joe Suzuki // Proceedings of the Ninth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-93). — San Francisco, CA: Morgan Kaufmann, 1993. — Pp. 266–273.

[12] Терентьев, А. Н. Эвристический метод построения байесовых сетей / А. Н. Терентьев, П. И. Бидюк // Математические машины и системы. —

2006. — № 3.

[13] Chow, C. I. Approximating discrete probability distributions with dependence trees / C. I. Chow, Sexior Member, C. N. Liu // IEEE Transactions on Information Theory. — 1968. — Vol. 14. — Pp. 462–467.

[14] Факторизация числа, проблемы ее теории [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habrahabr.ru/post/226395/. — Дата доступа: 22.03.2016.

[15] Computer Security Division - Computer Security Resource Center [Электронный ресурс]. — Электронные данные. — Режим доступа: https://nist.gov/post-free/6542/. — Дата доступа: 18.01.2016.

[16] Палицын, В. А. Технико-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В. А. Палицын. — Минск : БГУИР, 2006. — 76 с.

[17] Crundwell, F. K. Finance for engineers evaluation and funding of capital projects / F. K. Crundwell. — London : Springer, 2008. — 622 P.

ПРИЛОЖЕНИЕ А

```
(function(module) {

    "use strict";

    var Westley = require("./Westley.js"),
        Inigo = require("./InigoGenerator.js"),
        Flattener = require("./Flattener.js"),
        ManagedGroup = require("./ManagedGroup.js"),
        ManagedEntry = require("./ManagedEntry.js");

    var signing = require("../tools/signing.js"),
        rawSearching = require("../tools/searching-raw.js"),
        instanceSearching = require("../tools/searching-instance.js");

    /**
     * Find entries by searching properties/meta
     * @param {Archive} archive
     * @param {string} check Information to check (property/meta)
     * @param {string} key The key (property/meta-value) to search with
     * @param {RegExp|string} value The value to search for
     * @returns {Array.<ManagedEntry>}
     * @private
     * @static
     * @memberof Archive
     */
    function findEntriesByCheck(archive, check, key, value) {
        return instanceSearching.findEntriesByCheck(
            archive.getGroups(),
            function(entry) {
                var itemValue = (check === "property") ?
                    entry.getProperty(key) || "" :
                    entry.getMeta(key) || "";
                if (value instanceof RegExp) {
                    return value.test(itemValue);
                } else {
                    return itemValue.indexOf(value) >= 0;
                }
            }
        );
    }

    /**
     * The base Buttercup Archive class
     * @class Archive
     */
    var Archive = function() {
        var date = new Date(),
            ts = date.getFullYear() + "-" + (date.getMonth() + 1) + "-" + date.getDate();
        this._westley = new Westley();
        this._getWestley().execute(
            Inigo.create(Inigo.Command.Comment)
```

```
            .addArgument('Buttercup archive created (' + ts + ')')
            .generateCommand()
    );
    this._getWestley().execute(
        Inigo.create(Inigo.Command.Format)
            .addArgument(signing.getFormat())
            .generateCommand()
    );
};

/**
 * Whether or not this archive has a group with the given title.
 * @param {String} The group's title
 * @returns {true|false}
 * @deprecated Use findGroupsByTitle instead
 * @see findGroupsByTitle
 * @memberof Archive
 */
Archive.prototype.containsGroupWithTitle = function(groupTitle) {
    return this.findGroupsByTitle(groupTitle).length > 0;
};

/**
 * Create a new group
 * @param {string=} title The title for the group
 * @returns {ManagedGroup}
 * @memberof Archive
 */
Archive.prototype.createGroup = function(title) {
    var managedGroup = ManagedGroup.createNew(this);
    if (title) {
        managedGroup.setTitle(title);
    }
    return managedGroup;
};

/**
 * Find entries that match a certain meta property
 * @param {string} metaName The meta property to search for
 * @param {RegExp|string} value The value to search for
 * @returns {Array.<ManagedEntry>}
 * @memberof Archive
 */
Archive.prototype.findEntriesByMeta = function(metaName, value) {
    return findEntriesByCheck(this, "meta", metaName, value);
};

/**
 * Find all entries that match a certain property
 * @param {string} property The property to search with
 * @param {RegExp|string} value The value to search for
 * @returns {Array.<ManagedEntry>}
 * @memberof Archive
```

```
 */
Archive.prototype.findEntriesByProperty = function(property, value) {
    return findEntriesByCheck(this, "property", property, value);
};

/**
 * Find all groups within the archive that match a title
 * @param {RegExp|string} title The title to search for, either a string (contained
     within
 * a target group's title) or a RegExp to test against the title.
 * @returns {Array.<ManagedGroup>}
 * @memberof Archive
 */
Archive.prototype.findGroupsByTitle = function(title) {
    return instanceSearching.findGroupsByCheck(
        this.getGroups(),
        function(group) {
            if (title instanceof RegExp) {
                return title.test(group.getTitle());
            } else {
                return group.getTitle().indexOf(title) >= 0;
            }
        }
    );
};

/**
 * Find an entry by its ID
 * @param {String} The entry's ID
 * @returns {ManagedEntry|null}
 * @memberof Archive
 */
Archive.prototype.getEntryByID = function(entryID) {
    var westley = this._getWestley();
    var entryRaw = rawSearching.findEntryByID(westley.getDataset().groups, entryID);
    return (entryRaw === null) ? null : new ManagedEntry(this, entryRaw);
};

/**
 * Find a group by its ID
 * @param {String} The group's ID
 * @returns {ManagedGroup|null}
 * @memberof Archive
 */
Archive.prototype.getGroupByID = function(groupID) {
    var westley = this._getWestley();
    var groupRaw = rawSearching.findGroupByID(westley.getDataset().groups, groupID);
    return (groupRaw === null) ? null : new ManagedGroup(this, groupRaw);
};

/**
 * Get all groups (root) in the archive
 * @returns {ManagedGroup[]} An array of ManagedGroups
```

```
 * @memberof Archive
 */
Archive.prototype.getGroups = function() {
    var archive = this,
        westley = this._getWestley();
    return (westley.getDataset().groups || []).map(function(rawGroup) {
        return new ManagedGroup(archive, rawGroup);
    });
};

/**
 * Get the trash group
 * @returns {ManagedGroup|null}
 * @memberof Archive
 */
Archive.prototype.getTrashGroup = function() {
    var groups = this.getGroups();
    for (var i = 0, groupsLen = groups.length; i < groupsLen; i += 1) {
        if (groups[i].isTrash()) {
            return groups[i];
        }
    }
    return null;
};

/**
 * Perform archive optimisations
 * @memberof Archive
 */
Archive.prototype.optimise = function() {
    var flattener = new Flattener(this._getWestley());
    if (flattener.canBeFlattened()) {
        flattener.flatten();
    }
};

/**
 * Get the underlying Westley instance
 * @protected
 * @returns {Westley}
 * @memberof Archive
 */
Archive.prototype._getWestley = function() {
    return this._westley;
};

/**
 * Create an Archive with the default template
 * @returns {Archive} The new archive
 * @memberof Archive
 * @static
 */
Archive.createWithDefaults = function() {
```

```
        var archive = new Archive(),
            generalGroup = archive.createGroup("General"),
            trashGroup = archive
                .createGroup("Trash")
                    .setAttribute(ManagedGroup.Attributes.Role, "trash");
        return archive;
    };

    module.exports = Archive;

})(module);
```

ПРИЛОЖЕНИЕ Б

```
(function(module) {

    "use strict";

    var Inigo = require("./InigoGenerator.js"),
        ManagedEntry = require("./ManagedEntry.js"),
        encoding = require("../tools/encoding.js"),
        searching = require("../tools/searching-raw.js");

    /**
     * Managed group class
     * @class ManagedGroup
     * @param {Archive} archive The archive instance
     * @param {Object} remoteObj The remote object reference
     */
    var ManagedGroup = function(archive, remoteObj) {
        this._archive = archive;
        this._westley = archive._getWestley();
        this._remoteObject = remoteObj;
    };

    /**
     * Create a new entry with a title
     * @param {string=} title
     * @returns {ManagedEntry} The new entry
     * @memberof ManagedGroup
     */
    ManagedGroup.prototype.createEntry = function(title) {
        var managedEntry = ManagedEntry.createNew(this._getArchive(), this.getID());
        if (title) {
            managedEntry.setProperty("title", title);
        }
        return managedEntry;
    };

    /**
     * Create a child group
     * @param {string=} title Optionally set a title
     * @returns {ManagedGroup} The new child group
     * @memberof ManagedGroup
     */
    ManagedGroup.prototype.createGroup = function(title) {
        var group = ManagedGroup.createNew(this._getArchive(), this.getID());
        if (title) {
            group.setTitle(title);
        }
        return group;
    };

    /**
     * Delete the group
```

```
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.delete = function() {
    if (this.isTrash()) {
        throw new Error("Trash␣group␣cannot␣be␣deleted");
    }
    this._getWestley().execute(
        Inigo.create(Inigo.Command.DeleteGroup)
            .addArgument(this.getID())
            .generateCommand()
    );
    this._getWestley().pad();
    delete this._westley;
    delete this._remoteObject;
};

/**
 * Delete an attribute
 * @param {string} attr The name of the attribute
 * @returns {ManagedGroup} Returns self
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.deleteAttribute = function(attr) {
    this._getWestley().execute(
        Inigo.create(Inigo.Command.DeleteGroupAttribute)
            .addArgument(this.getID())
            .addArgument(attr)
            .generateCommand()
    );
    this._getWestley().pad();
    return this;
};

/**
 * Get an attribute
 * @param {string} attributeName The name of the attribute
 * @returns {string|undefined} Returns the attribute or undefined if not found
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.getAttribute = function(attributeName) {
    var raw = this._getRemoteObject();
    return raw.attributes && raw.attributes.hasOwnProperty(attributeName) ?
        raw.attributes[attributeName] : undefined;
};

/**
 * Get the entries within the group
 * @returns {Array.<ManagedEntry>}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.getEntries = function() {
    var archive = this._getArchive();
    return (this._getRemoteObject().entries || []).map(function(rawEntry) {
```

```
        return new ManagedEntry(archive, rawEntry);
    });
};

/**
 * Get the groups within the group
 * @returns {Array.<ManagedGroup>}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.getGroups = function() {
    var archive = this._getArchive();
    return (this._getRemoteObject().groups || []).map(function(rawGroup) {
        return new ManagedGroup(archive, rawGroup);
    });
};

/**
 * Get the group ID
 * @returns {string}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.getID = function() {
    return this._getRemoteObject().id;
};

/**
 * Get the group title
 * @returns {string}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.getTitle = function() {
    return this._getRemoteObject().title || "";
};

/**
 * Check if the current group is used for trash
 * @returns {boolean}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.isTrash = function() {
    return this.getAttribute(ManagedGroup.Attributes.Role) === "trash";
};

/**
 * Move the group into another
 * @param {ManagedGroup} group The target group (new parent)
 * @returns {ManagedGroup} Returns self
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.moveToGroup = function(group) {
    if (this.isTrash()) {
        throw new Error("Trash␣group␣cannot␣be␣moved");
    }
```

```
    var targetID = group.getID();
    this._getWestley().execute(
        Inigo.create(Inigo.Command.MoveGroup)
            .addArgument(this.getID())
            .addArgument(targetID)
            .generateCommand()
    );
    this._getWestley().pad();
    return this;
};

/**
 * Set an attribute
 * @param {string} attributeName The name of the attribute
 * @param {string} value The value to set
 * @returns {ManagedGroup} Returns self
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.setAttribute = function(attributeName, value) {
    this._getWestley().execute(
        Inigo.create(Inigo.Command.SetGroupAttribute)
            .addArgument(this.getID())
            .addArgument(attributeName)
            .addArgument(value)
            .generateCommand()
    );
    this._getWestley().pad();
    return this;
};

/**
 * Set the group title
 * @param {string} title The title of the group
 * @returns {ManagedGroup} Returns self
 */
ManagedGroup.prototype.setTitle = function(title) {
    this._getWestley().execute(
        Inigo.create(Inigo.Command.SetGroupTitle)
            .addArgument(this.getID())
            .addArgument(title)
            .generateCommand()
    );
    this._getWestley().pad();
    return this;
};

/**
 * Export group to object
 * @returns {Object}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype.toObject = function() {
    // @todo use object cloning
```

```
    var attributes = {},
        groupAttributes = this._remoteObject.attributes || {};
    for (var attrKey in groupAttributes) {
        if (groupAttributes.hasOwnProperty(attrKey)) {
            attributes[attrKey] = groupAttributes[attrKey];
        }
    }
    return {
        id: this.getID(),
        title: this.getTitle(),
        attributes: attributes
    };
};

ManagedGroup.prototype.toString = function() {
    return JSON.stringify(this.toObject());
};

/**
 * Get the archive instance reference
 * @protected
 * @returns {Archive}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype._getArchive = function() {
    return this._archive;
};

/**
 * Get the remotely-managed object (group)
 * @protected
 * @returns {Object}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype._getRemoteObject = function() {
    return this._remoteObject;
};

/**
 * Get the delta managing instance for the archive
 * @protected
 * @returns {Westley}
 * @memberof ManagedGroup
 */
ManagedGroup.prototype._getWestley = function() {
    return this._westley;
};

ManagedGroup.Attributes = Object.freeze({
    Role: "bc_group_role"
});

/**
```

```
     * Create a new ManagedGroup with a delta-manager and parent group ID
     * @static
     * @memberof ManagedGroup
     * @param {Archive} archive
     * @param {string=} parentID The parent group ID (default is root)
     * @returns {ManagedGroup} A new group
     */
    ManagedGroup.createNew = function(archive, parentID) {
        parentID = parentID || "0";
        var id = encoding.getUniqueID(),
            westley = archive._getWestley();
        westley.execute(
            Inigo.create(Inigo.Command.CreateGroup)
                .addArgument(parentID)
                .addArgument(id)
                .generateCommand()
        );
        var group = searching.findGroupByID(westley.getDataset().groups, id);
        return new ManagedGroup(archive, group);
    };

    module.exports = ManagedGroup;

})(module);
```

ПРИЛОЖЕНИЕ В

```
import assert = require('assert');
import Q = require('q');

import collectionutil = require('./base/collectionutil');
import dateutil = require('./base/dateutil');
import err_util = require('./base/err_util');
import event_stream = require('./base/event_stream');
import item_merge = require('./item_merge');
import item_store = require('./item_store');
import key_agent = require('./key_agent');
import logging = require('./base/logging');

export class SyncError extends err_util.BaseError {
        constructor(message: string, sourceErr?: Error) {
                super(message, sourceErr);
        }
}

let syncLog = new logging.BasicLogger('sync');
syncLog.level = logging.Level.Warn;

const REMOTE_STORE = 'cloud';

/** Returns true if two date/times from Item.updatedAt should
  * be considered equal for the purpose of sync.
  *
  * This function accounts for the fact that the resolution
  * of timestamps varies depending on the store - eg.
  * the Agile Keychain format uses timestamps with only
  * second-level resolution whereas local_store.Store supports
  * millisecond-resolution timestamps.
  */
export function itemUpdateTimesEqual(a: Date, b: Date) {
        return dateutil.unixTimestampFromDate(a) ==
                dateutil.unixTimestampFromDate(b);
}

export enum SyncState {
        /** Sync is not currently in progress */
        Idle,

        /** Sync is enumerating changed items */
        ListingItems,

        /** Sync is fetching and updating changed items */
        SyncingItems
}

export interface SyncProgress {
        state: SyncState;
```

```
        /** Count of items that have been synced. */
        updated: number;

        /** Count of items that failed to sync. */
        failed: number;

        /** Total number of changed items to sync. */
        total: number;

        /** Number of items actively being synced. */
        active: number;
}

enum ItemSyncState {
        Unchanged,
        Updated,
        Deleted
}

interface SyncItem {
        localItem: item_store.ItemState;
        localState: ItemSyncState;
        remoteItem: item_store.ItemState;
        remoteState: ItemSyncState;
}

/** Interface for syncing encryption keys and items between
  * a cloud-based store and a local cache.
  */
export interface Syncer {
        onProgress: event_stream.EventStream<SyncProgress>;

        /** Sync encryption keys from the remote store to the local one.
          * This does not require the remote store to be unlocked.
          */
        syncKeys(): Q.Promise<void>;

        /** Sync items between the local and remote stores.
          * Returns a promise which is resolved when the current sync completes.
          *
          * Syncing items requires both local and remote stores
          * to be unlocked first.
          */
        syncItems(): Q.Promise<SyncProgress>;
}

/** Syncer implementation which syncs changes between an item_store.Store
  * representing a remote store and a local store.
  */
export class EventEmittor implements Syncer {
        private localStore: item_store.SyncableStore;
        private cloudStore: item_store.Store;
```

```
// queue of items left to sync
private syncQueue: SyncItem[];
// progress of the current sync
private syncProgress: SyncProgress;
// promise for the result of the
// current sync task or null if no sync
// is in progress
private currentSync: Q.Deferred<SyncProgress>;

onProgress: event_stream.EventStream<SyncProgress>;

constructor(localStore: item_store.SyncableStore, cloudStore: item_store.Store) {
        this.localStore = localStore;
        this.cloudStore = cloudStore;
        this.onProgress = new event_stream.EventStream<SyncProgress>();
        this.syncQueue = [];
}

syncKeys(): Q.Promise<void> {
        let keys = this.cloudStore.listKeys();

        // sync the password hint on a best-effort basis.
        // If no hint is available, display a placeholder instead.
        let hint = this.cloudStore.passwordHint().then(hint => {
                return hint;
        }).catch(err => {
                return '';
        });

        return Q.all([keys, hint]).then((keysAndHint) => {
                var keys = <key_agent.Key[]>keysAndHint[0];
                var hint = <string>keysAndHint[1];
                return this.localStore.saveKeys(keys, hint);
        });
}

syncItems(): Q.Promise<SyncProgress> {
        if (this.currentSync) {
                // if a sync is already in progress, complete the current sync
                    first.
                // This should queue up a new sync to complete once the current one
                     finishes.
                return this.currentSync.promise;
        }
        syncLog.info('Starting sync');

        var result = Q.defer<SyncProgress>();
        this.currentSync = result;
        this.currentSync.promise.then(() => {
                syncLog.info('Sync completed');
                this.currentSync = null;
        }).catch(err => {
                syncLog.error('Sync failed', err.toString());
```

```
        this.currentSync = null;
});

this.syncProgress = {
        state: SyncState.ListingItems,
        active: 0,
        updated: 0,
        failed: 0,
        total: 0
};

this.onProgress.listen(() => {
        if (this.syncProgress.state == SyncState.SyncingItems) {
                var processed = this.syncProgress.updated + this.
                    syncProgress.failed;
                syncLog.info({
                        updated: this.syncProgress.updated,
                        failed: this.syncProgress.failed,
                        total: this.syncProgress.total
                });

                if (processed == this.syncProgress.total) {
                        this.syncProgress.state = SyncState.Idle;
                        this.notifyProgress();
                        result.resolve(this.syncProgress);
                } else {
                        this.syncNextBatch();
                }
        }
}, 'sync-progress');
this.notifyProgress();

let localItems = this.localStore.listItemStates();
let remoteItems = this.cloudStore.listItemStates();
let lastSyncRevisions = this.localStore.lastSyncRevisions(REMOTE_STORE);

Q.all([localItems, remoteItems, lastSyncRevisions]).then(itemLists => {
        let localItems = <item_store.ItemState[]>itemLists[0];
        let remoteItems = <item_store.ItemState[]>itemLists[1];
        let lastSyncedRevisions = <Map<string, item_store.RevisionPair>>
            itemLists[2];

        syncLog.info('%d items in local store, %d in remote store',
            localItems.length, remoteItems.length);

        let allItems: { [index: string]: boolean } = {};

        let localItemMap = collectionutil.listToMap(localItems, item => {
                allItems[item.uuid] = true;
                return item.uuid;
        });
        let remoteItemMap = collectionutil.listToMap(remoteItems, item => {
                allItems[item.uuid] = true;
```

```
                        return item.uuid;
                });

                Object.keys(allItems).forEach(uuid => {
                        let remoteItem = remoteItemMap.get(uuid);
                        let localItem = localItemMap.get(uuid);
                        let lastSyncedRevision = lastSyncedRevisions.get(uuid);

                        this.enqueueItemForSyncIfChanged(localItem, remoteItem,
                            lastSyncedRevision);
                });

                syncLog.info('found %d items to sync', this.syncQueue.length);
                this.syncProgress.state = SyncState.SyncingItems;
                this.notifyProgress();
        }).catch(err => {
                syncLog.error('Failed to list items in local or remote stores', err
                    .stack);
                result.reject(err);

                this.syncProgress.state = SyncState.Idle;
                this.notifyProgress();
        });

        this.syncNextBatch();

        return this.currentSync.promise;
}

private enqueueItemForSyncIfChanged(localItem: item_store.ItemState,
        remoteItem: item_store.ItemState,
        lastSyncedRevision?: item_store.RevisionPair) {

        let uuid = localItem ? localItem.uuid : remoteItem.uuid;
        let remoteState = ItemSyncState.Unchanged;
        let localState = ItemSyncState.Unchanged;

        if (localItem) {
                if (localItem.deleted) {
                        if (lastSyncedRevision) {
                                localState = ItemSyncState.Deleted;
                                syncLog.info('item %s deleted locally', uuid);
                        }
                } else if (lastSyncedRevision) {
                        if (localItem.revision !== lastSyncedRevision.local) {
                                localState = ItemSyncState.Updated;
                                syncLog.info('item %s updated locally', uuid);
                        }
                } else {
                        localState = ItemSyncState.Updated;
                        syncLog.info('item %s added locally');
                }
        }
```

```
        if (remoteItem) {
                if (remoteItem.deleted) {
                        if (lastSyncedRevision) {
                                remoteState = ItemSyncState.Deleted;
                                syncLog.info('item %s deleted in cloud', uuid);
                        }
                } else if (lastSyncedRevision) {
                        if (remoteItem.revision !== lastSyncedRevision.external) {
                                remoteState = ItemSyncState.Updated;
                                syncLog.info('item %s updated in cloud', uuid);
                        }
                } else {
                        remoteState = ItemSyncState.Updated;
                        syncLog.info('item %s added in cloud', uuid);
                }
        }

        if (localState !== ItemSyncState.Unchanged ||
                remoteState !== ItemSyncState.Unchanged) {
                this.syncQueue.push({
                        localItem: localItem,
                        localState: localState,
                        remoteItem: remoteItem,
                        remoteState: remoteState
                });
                ++this.syncProgress.total;
        }
}

// sync the next batch of items. This adds items
// to the queue to sync until the limit of concurrent items
// being updated at once reaches a limit.
//
// When syncing with a store using the Agile Keychain format
// and Dropbox, there is one file to fetch per-item so this
// batching nicely maps to network requests that we'll need
// to make. If we switch to the Cloud Keychain format (or another
// format) in future which stores multiple items per file,
// the concept of syncing batches of items may no longer
// be needed or may need to work differently.
private syncNextBatch() {
        var SYNC_MAX_ACTIVE_ITEMS = 10;
        while (this.syncProgress.active < SYNC_MAX_ACTIVE_ITEMS &&
                this.syncQueue.length > 0) {
                var next = this.syncQueue.shift();
                this.syncItem(next);
        }
}

private notifyProgress() {
        this.onProgress.publish(this.syncProgress);
}
```

```
// create a tombstone item to represent an item
// which has been deleted locally or in the cloud during sync
private createTombstone(store: item_store.Store, uuid: string): item_store.
    ItemAndContent {
        let item = new item_store.Item(store, uuid);
        item.typeName = item_store.ItemTypes.TOMBSTONE;
        item.updatedAt = new Date();
        return {
                item: item,
                content: null
        };
}

private syncItem(item: SyncItem) {
        ++this.syncProgress.active;

        var itemDone = (err?: Error) => {
                --this.syncProgress.active;
                if (err) {
                        ++this.syncProgress.failed;
                } else {
                        ++this.syncProgress.updated;
                }
                this.notifyProgress();
        };

        // fetch content for local and remote items and the last-synced
        // version of the item in order to perform a 3-way merge
        let uuid = item.localItem ? item.localItem.uuid : item.remoteItem.uuid;
        let localItemContent: Q.Promise<item_store.ItemAndContent>;
        let remoteItemContent: Q.Promise<item_store.ItemAndContent>;

        if (item.localItem) {
                if (item.localItem.deleted) {
                        localItemContent = Q(this.createTombstone(this.localStore,
                            uuid));
                } else {
                        localItemContent = this.localStore.loadItem(uuid);
                }
        }

        if (item.remoteItem) {
                if (item.remoteItem.deleted) {
                        remoteItemContent = Q(this.createTombstone(this.cloudStore,
                            uuid));
                } else {
                        remoteItemContent = this.cloudStore.loadItem(uuid);
                }
        }

        let contents = Q.all([localItemContent, remoteItemContent]);
        contents.then((contents: [item_store.ItemAndContent, item_store.
```

```
            ItemAndContent]) => {
                // merge changes between local/remote store items and update the
                // last-synced revision
                let localItem = contents[0];
                let remoteItem = contents[1];
                this.mergeAndSyncItem(localItem, item.localState, remoteItem, item.
                    remoteState)
                .then(() => {
                        syncLog.info('Synced changes for item %s', uuid);
                        itemDone();
                }).catch((err: Error) => {
                        syncLog.error('Syncing item %s failed:', uuid, err);
                        var itemErr = new SyncError(`Failed to save updates for item
                             ${uuid}`, err);
                        itemDone(itemErr);
                });
        }).catch(err => {
                syncLog.error('Retrieving updates for %s failed:', uuid, err);
                var itemErr = new SyncError(`Failed to retrieve updated item ${uuid
                    }`, err);
                itemDone(itemErr);
        });
}

// returns the item and content for the last-synced version of an item,
// or null if the item has not been synced before
private getLastSyncedItemRevision(uuid: string): Q.Promise<item_store.
    ItemAndContent> {
        return this.localStore.getLastSyncedRevision(uuid, REMOTE_STORE).then(
            revision => {
                if (revision) {
                        return this.localStore.loadItem(uuid, revision.local);
                } else {
                        return null;
                }
        });
}

// given an item from the local and remote stores, one or both of which have
    changed
// since the last sync, and the last-synced version of the item, merge
// changes and save the result to the local/remote store as necessary
//
// When the save completes, the last-synced revision is updated in
// the local store
private mergeAndSyncItem(localItem: item_store.ItemAndContent,
        localState: ItemSyncState,
        remoteItem: item_store.ItemAndContent,
        remoteState: ItemSyncState) {
        assert(localItem || remoteItem, 'neither local nor remote item specified');

        let remoteRevision: string;
        if (remoteItem && !remoteItem.item.isTombstone()) {
```

```
        remoteRevision = remoteItem.item.revision;
        assert(remoteRevision, 'item does not have a remote revision');
}

let updatedStoreItem: item_store.Item;
let saved: Q.Promise<void>;

// revision of the item which was saved
let newLocalRevision: string;

if (localState === ItemSyncState.Updated && remoteState === ItemSyncState.
    Updated) {
        // item updated both locally and in the cloud, merge changes
        assert(localItem);
        assert(remoteItem);
        syncLog.info('merging local and remote changes for item %s',
            localItem.item.uuid);

        let mergedStoreItem: item_store.ItemAndContent;
        saved = this.getLastSyncedItemRevision(localItem.item.uuid).then(
            lastSynced => {
                mergedStoreItem = item_merge.merge(localItem, remoteItem,
                    lastSynced);
                mergedStoreItem.item.updateTimestamps();

                let mergedRemoteItem = item_store.cloneItem(mergedStoreItem,
                     mergedStoreItem.item.uuid);

                return Q.all([
                        this.localStore.saveItem(mergedStoreItem.item,
                            item_store.ChangeSource.Sync),
                        this.cloudStore.saveItem(mergedRemoteItem.item,
                            item_store.ChangeSource.Sync)
                ]);
        }).then(() => {
                assert(mergedStoreItem.item.revision, 'merged local item
                    does not have a revision');
                assert.notEqual(mergedStoreItem.item.revision, localItem.
                    item.revision);

                newLocalRevision = mergedStoreItem.item.revision;
                updatedStoreItem = mergedStoreItem.item;
        });

} else if (localState !== ItemSyncState.Unchanged) {
        // item added/updated/removed locally
        syncLog.info('syncing item %s from local -> cloud', localItem.item.
            uuid);
        let clonedItem = item_store.cloneItem(localItem, localItem.item.
            uuid).item;
        newLocalRevision = localItem.item.revision;
        updatedStoreItem = localItem.item;
        saved = this.cloudStore.saveItem(clonedItem, item_store.
```

```
                    ChangeSource.Sync).then(() => {
                        remoteRevision = clonedItem.revision;
                });
            } else if (remoteState !== ItemSyncState.Unchanged) {
                // item added/updated/removed in cloud
                syncLog.info('syncing item %s from cloud -> local', remoteItem.item
                    .uuid);
                let clonedItem = item_store.cloneItem(remoteItem, remoteItem.item.
                    uuid).item;
                saved = this.localStore.saveItem(clonedItem, item_store.
                    ChangeSource.Sync).then(() => {
                        assert(clonedItem.revision, 'item cloned from remote store
                            does not have a revision');
                        newLocalRevision = clonedItem.revision;
                        updatedStoreItem = clonedItem;
                });
            }

            return saved.then(() => {
                syncLog.info('setting last synced revisions for %s to %s, %s',
                    updatedStoreItem.uuid, newLocalRevision, remoteRevision);

                let revisions: item_store.RevisionPair;
                if (!updatedStoreItem.isTombstone()) {
                    assert(newLocalRevision, 'saved item does not have a
                        revision');
                    revisions = { local: newLocalRevision, external:
                        remoteRevision };
                }
                return this.localStore.setLastSyncedRevision(updatedStoreItem,
                    REMOTE_STORE, revisions);
            });
        }
    }
```

ВЕДОМОСТЬ ДОКУМЕНТОВ