

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

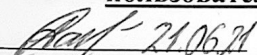
БАКАЛАВРСКАЯ РАБОТА

на тему

Программное обеспечение администрирования учётных данных

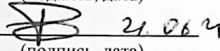
пользователей

Студент


(подпись, дата)

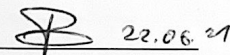
Самсонкин Андрей Сергеевич
(ФИО полностью)

Руководитель


(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

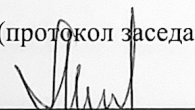
Нормоконтролер


(подпись, дата)

Такташкин Д.В.
(фамилия, инициалы)

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

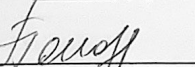
Заведующий кафедрой


(подпись)

Макарычев П.П.
(фамилия, инициалы)

Работа защищена с отметкой 5 (протокол заседания ГЭК от 28.06.21 № 4)

Секретарь ГЭК


(подпись)

Попова Н.А.
(фамилия, инициалы)

Пенза, 2021

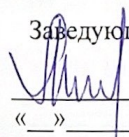
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»

Заведующий кафедрой

 П.П. Макарычев
«__» _____ 2021 г

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент: Самсонкин Андрей Сергеевич.17ВИ1 факультета ФВТ
2. Тема работы: программное обеспечение администрирования учётных данных пользователей

Тема утверждена приказом ПГУ _____ от "___" _____ 2021 г.

3. Руководитель работы к.т.н доцент. Такташкин Д.В

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: сохранение и чтение учётных записей пользователей

Функции системы:

1. Задание программы с помощью специального языка

2. Проверка программы на правильность

3. Выполнение программы

Технология разработки: ООП. Кроссплатформенная разработка

Язык программирования: Kotlin

Среда программирования: JetBrains IntelliJ IDEA

5. Перечень подлежащих разработке вопросов

1. Анализ требований на разработку

2. Проектирование системы

3. Реализация разрабатываемой системы

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области и требований на разработку	5%	25.09.20 16.10.20	
2.	Проектирование программных средств	10%	16.10.20 10.11.20	
3.	Разработка программных средств	30%	10.11.20 28.03.21	
4.	Тестирование программных средств	20%	28.03.21 18.04.21	
5.	Оформление пояснительной записки	35%	18.04.21 25.05.21	

Дата выдачи задания 28.04.2021

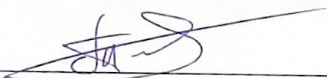
Руководитель бакалаврской работы
(подпись, дата)



Задание к исполнению принял студент
(подпись, дата)

 21.06.21

Работу к защите допустить
Декан факультета
(подпись, дата)



5. Перечень подлежащих разработке вопросов

1. Анализ требований на разработку

2. Проектирование системы

3. Реализация разрабатываемой системы

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области и требований на разработку	5%	25.09.20 16.10.20	
2.	Проектирование программных средств	10%	16.10.20 10.11.20	
3.	Разработка программных средств	30%	10.11.20 28.03.21	
4.	Тестирование программных средств	20%	28.03.21 18.04.21	
5.	Оформление пояснительной записки	35%	18.04.21 25.05.21	

Дата выдачи задания 28.04.2021

Руководитель бакалаврской работы

(подпись, дата)

Задание к исполнению принял студент

(подпись, дата)

Работу к защите допустить

Декан факультета

(подпись, дата)

Реферат

Пояснительная записка содержит 69 листов, 28 рисунков, 13 использованных источников, 3 таблицы и 3 приложения.

KOTLIN MULTIPLATFORM, INTELLIJ IDEA, АДМИНИСТРИРОВАНИЕ, УЧЁТНЫЕ ЗАПИСИ, КРОССПЛАТФОРМЕННАЯ РАЗРАБОТКА, ИНТРЕФЕЙС.

Объектом исследования являются программные обеспечения администрирования учётных данных пользователей.

Предмет исследования – технология кроссплатформенной разработки программного обеспечения администрирования учётных данных пользователей.

Цель работы – создание программного обеспечения администрирования учётных данных пользователей для улучшения безопасности в интернете и удобства пользования.

Технология разработки: ООП, Kotlin MPP, Kotlin.

Результаты работы: спроектировано, разработано, отлажено и протестировано программное обеспечение администрирования учётных данных пользователей.

Область применения: сохранение и заполнение учётных записей пользователя.

					ПГУ 09.03.02 – 6КП171.12 ПЗ								
Изм.	Лист	№ докум.	Подпись	Дата	Программное обеспечение администрирования учётных данных пользователей	Лит.			Лист		Листов		
Разраб.		Самсонкин А.С.		21.06.17					4		69		
Провер.		Такташкин Д.В.		21.06.17									
Реценз.													
И. Контр.		Такташкин Д.В.		21.06.17									
И. тв.											гр. 17ВИ1		

Содержание

Введение	6
1 Аналитическая часть	7
1.1 Анализ менеджеров паролей	7
1.1.1 Обзор Dashlane	7
1.1.2 Обзор LastPass	9
1.1.3 Обзор KeePass	10
1.1.4 Обзор Google Password Manager.....	11
1.1.5 Обзор Apple Keychain.....	11
1.2 Анализ требований к программе управления учётными данными.....	12
1.2.1 Функциональные требования	12
1.3.2 Нефункциональные требования	14
2 Проектирование системы управления учётными данными.....	15
2.1 Проектирование интерфейса управления учётными данными	15
2.2 Структура программы управления учётными данными	19
2.3 Технологический стек	22
3 Реализация ядра управления учётными данными	25
3.1 Программная реализация	25
3.2 Тестирование ядра управления учётными данными	35
3.3 Расчёт метрик кода системы управления учётными данными.....	37
3.4 Планирование разработки и оценка бюджета.....	39
Заключение	43
Список использованных источников	44
Приложение А. Глоссарий	46
Приложение Б. Диаграммы программы	48
Приложение В. Код программы и модульных тестов.....	51

Введение

В современном мире информационные технологии занимают все большее пространство во всех сферах жизни общества. Интернет стал неотъемлемой частью почти каждого современного человека. Огромное количество интернет-ресурсов требуют аутентификацию: электронная почта, социальные сети, веб-форумы, платёжные системы, интернет-магазины и другие ресурсы. Самый распространённый способ аутентификации на сегодняшний день является форма логин-пароль. Чтобы обеспечить безопасность, необходимо для каждого ресурса использовать уникальный длинный и сложный пароль. Когда количество таких ресурсов начинает превышать 2 – 3, возникает проблема хранения таких паролей.

Большинство людей использует одинаковые пароли для своих ресурсов, подвергаясь риску взлома, ведь если на одном ресурсе произойдёт утечка, то злоумышленники получат доступ ко всем ресурсам пользователя.

Одно из решений данной проблемы является использования биометрии для аутентификации. Она уже используется в большинстве мобильных устройств и крайне эффективна. Но этот способ аутентификации сложно реализуем и обычно подкрепляется кодом или паролем, что сводит на нет эффективность данного метода.

Другим эффективным и одним из самых распространённых решений является менеджер паролей. Они снимают с пользователя необходимость запоминания учётных данных для каждого ресурса, генерируют уникальный и сложный пароль. Таким образом пользователю необходимо помнить, как минимум один пароль.

К минимальным функциям подобных программ обычно относятся:

- хранения учётных данных;
- поиск учётных данных;
- шифрование учётных данных.

Все определения, используемые в данной выпускной квалификационной работе, находятся в приложении А.

1 Аналитическая часть

Многие специалисты информационной безопасности приводят рекомендации по созданию паролей. Самые базовые из них – использовать длинный, сложный и уникальный пароль для каждого интернет-ресурса. Держать в голове такие учётные данные для всех ресурсов практически невозможно. А писать на листке или в текстовом документе неудобно и небезопасно [1].

Менеджер паролей – это база данных, которая хранит учётные данные пользователя. Это не просто программа для удобства, а необходимый инструмент для защиты учетных данных, а в некоторых случаях и номеров банковских карт [2].

Таким образом менеджеры паролей необходимы практически любому пользователю, кто так или иначе пользуется интернет-ресурсами.

1.1 Анализ менеджеров паролей

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

1.1.1 Обзор Dashlane

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

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

Dashlane использует мастер-пароль для шифрования сохранённых паролей на устройстве [3].

На рисунке 1 представлен интерфейс программы.

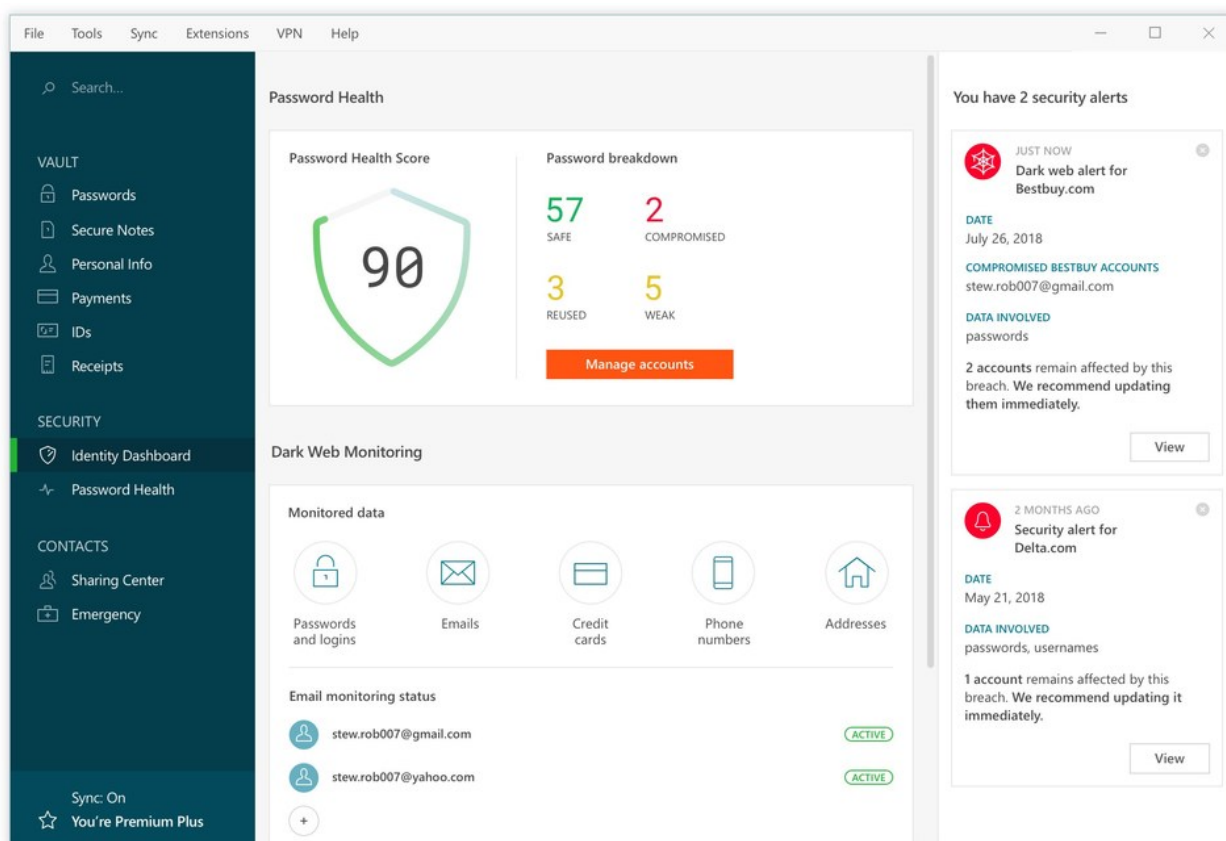


Рисунок 1 – Интерфейс Dashlane

Dashlane поддерживает такие функции, как:

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

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

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

1.1.2 Обзор LastPass

LastPass — это условно-бесплатный кроссплатформенный проприетарный менеджер паролей, разработанный компанией LogMeIn. Он поставляется как плагин для браузеров Internet Explorer, Google Chrome, Mozilla Firefox, Opera, Safari или как приложение для macOS, iOS, Android [4].

LastPass использует алгоритмы AES256 для шифрования и PBKDF2 SHA-256 для хеширования [4].

Интерфейс обозреваемой программы представлен на рисунке 2.

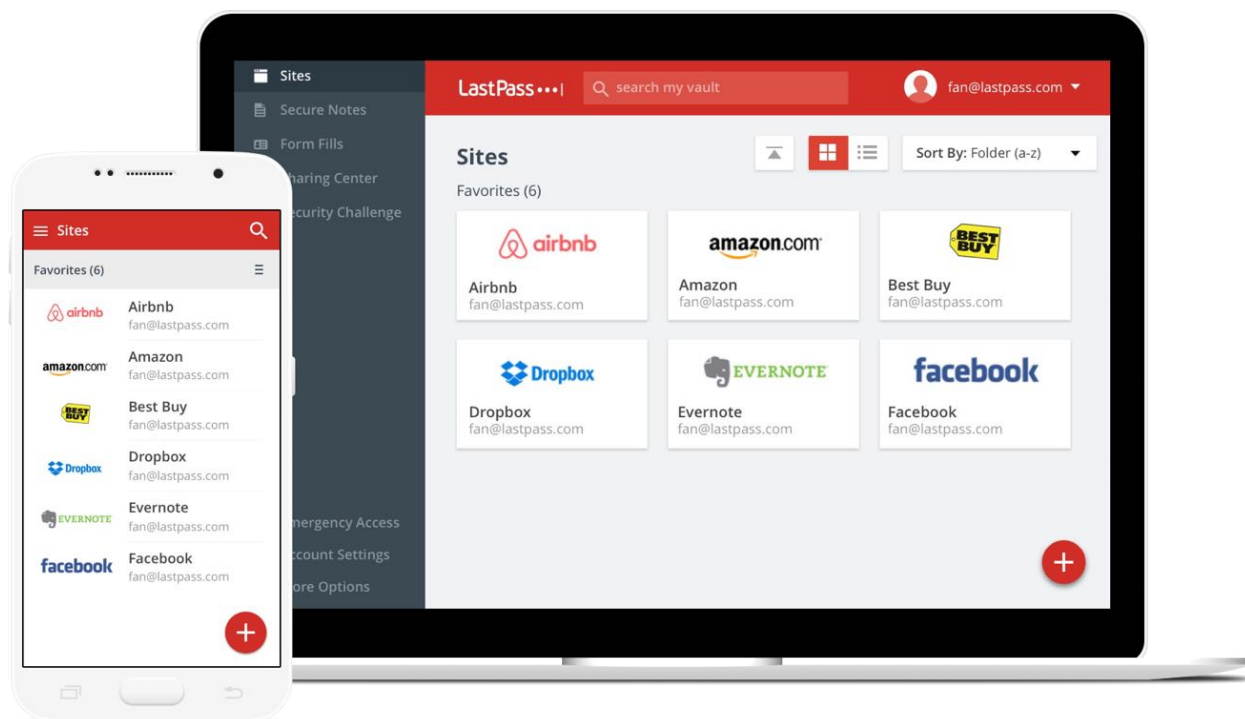


Рисунок 2 – Интерфейс LastPass

Возможности LastPass:

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

- зашифрованные заметки;
- зашифрованное хранилище данных.

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

За время жизни программы, было обнаружено большое количество уязвимостей и несколько утечек, что подрывает доверие к продукту [5].

1.1.3 Обзор KeePass

KeePass – это бесплатный кроссплатформенный менеджер паролей с открытым исходным кодом, разработанный Домиником Рейчелом. Он поставляется как приложение для основных настольных и мобильных платформ. На рисунке 3 представлен интерфейс программы.

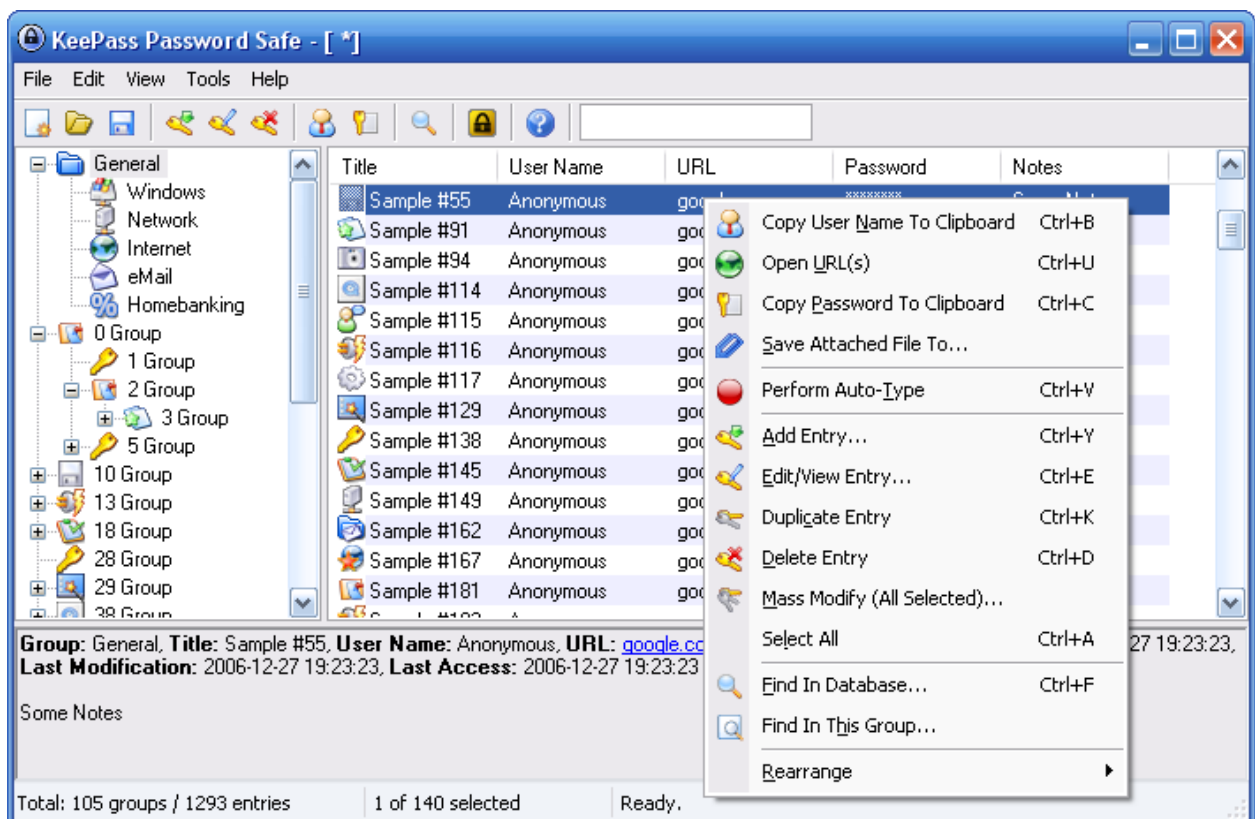


Рисунок 3 – Интерфейс KeePass

KeePass является портативным приложением для настольных платформ, написанным на C#. KeePass предлагает крайне широкий функционал

абсолютно бесплатно. Он имеет открытый исходный код. Отказ от хранения информации облаком увеличивает сохранность паролей.

Недостатком KeePass является то, что этот продукт рассчитан на более опытных пользователей, по сравнению с аналогами. Множество функций, которые упрощают пользование программой не реализованы [6].

1.1.4 Обзор Google Password Manager

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

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

1.1.5 Обзор Apple Keychain

Apple Keychain – это бесплатный менеджер паролей с открытым исходным кодом. Он работает только на устройствах от Apple. Это решение по умолчанию включает двухфакторную аутентификацию, как часть входа в аккаунт iCloud. На рисунке 4 представлен интерфейс программы.

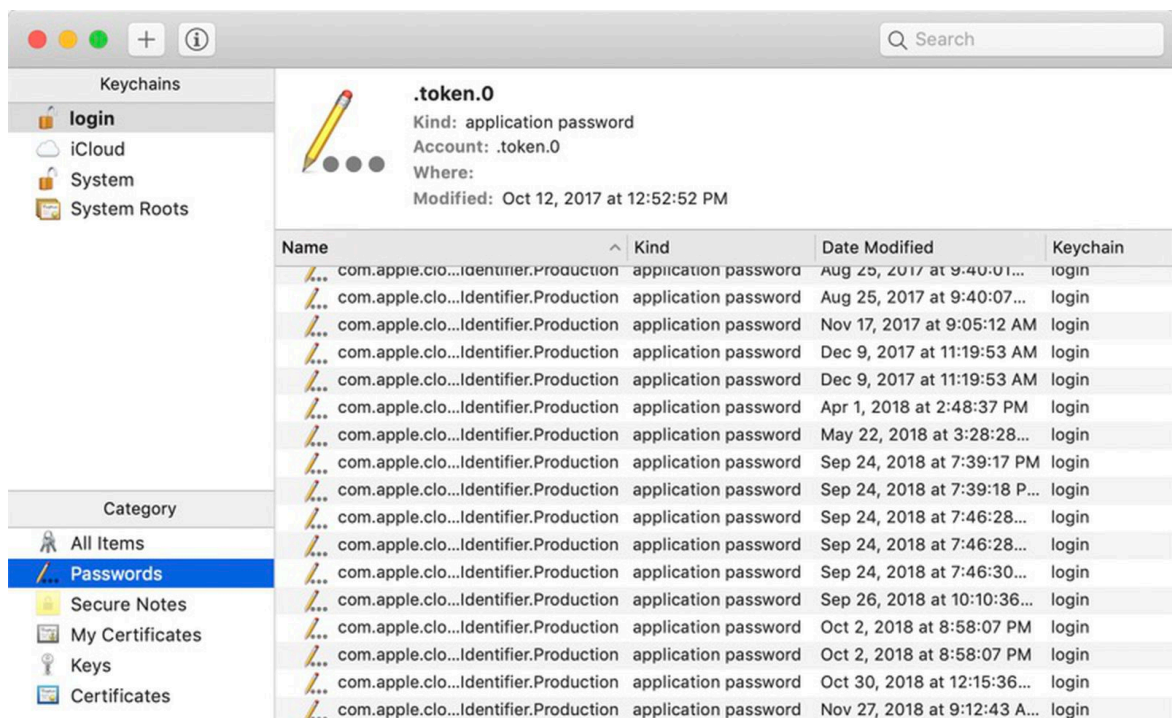


Рисунок 4 – Интерфейс Apple Keychain

Apple Keychain использует Triple DES для шифрования данных на устройстве.

Преимуществами этого решения. Apple имеет хорошую репутацию в сфере информационной безопасности. Keychain отлично работает на всех устройствах Apple. Triple DES является менее популярным алгоритмом шифрования [7].

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

В результате исследований были рассмотрены основные программы для хранения паролей и произведен их сравнительный анализ. Результаты приведены в таблице 1.

Таблица 1 – Сравнительный анализ программ для администрирования учётных записей пользователей

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

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

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

1.2 Анализ требований к программе управления учётными данными

1.2.1 Функциональные требования

Программа должна выполнять следующий функционал:

- задавать мастер-пароль хранилищу;
- сохранять учётные данные;
- обновлять учётные данные;
- удалять учётные данные;

- проверять мастер-пароль при доступе к хранилищу;
- очищать учётные данные.

Для наглядного описания поведения разрабатываемой системы была составлена диаграмма вариантов использования (рисунок 5), состоящая из актеров и выполняемых ими прецедентов.

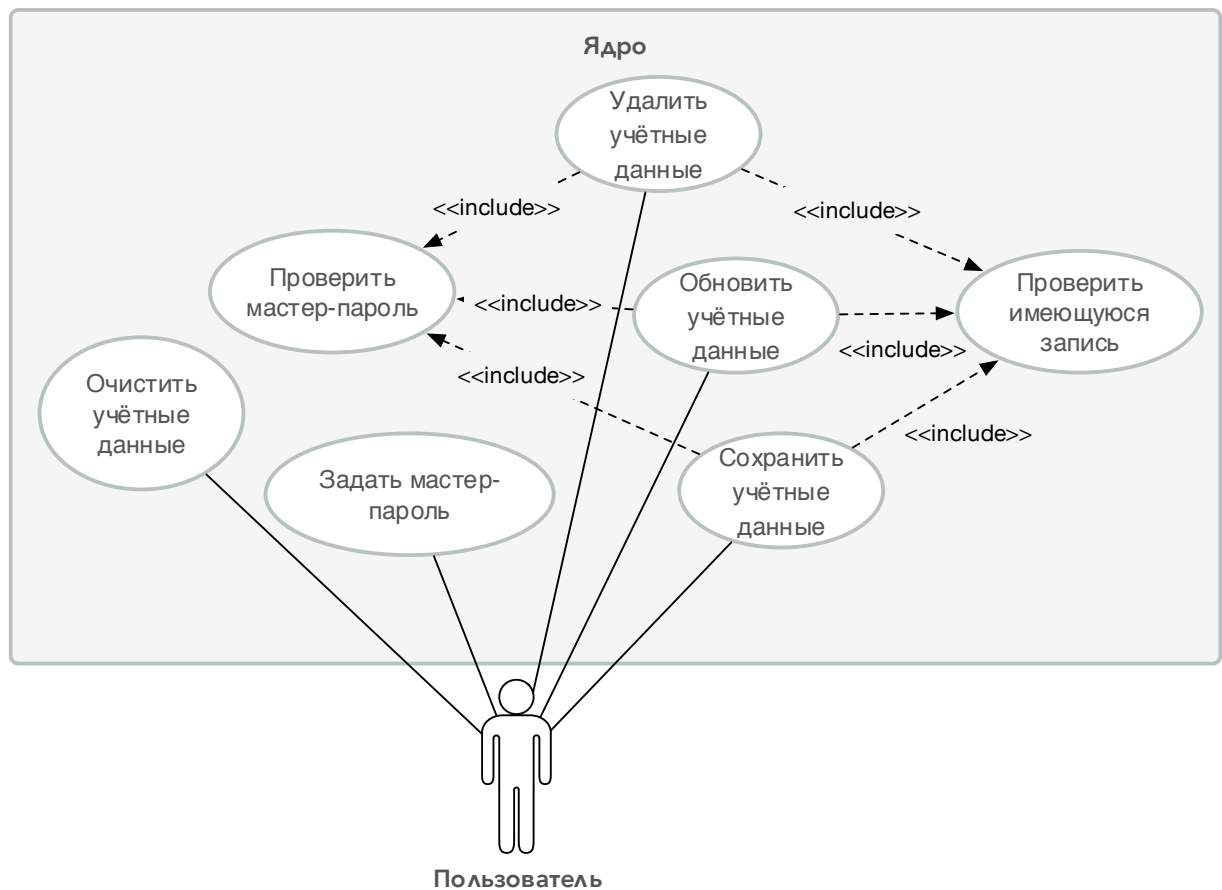


Рисунок 5 – Диаграмма вариантов использования

Подробное описание одного прецедента сохранения учётных данных пользователя представлен в таблице 2.

Таблица 2 – Вариант сохранения учётных данных

Прецедент: Сохранить учётные данные
ID: 1
Краткое описание: Сохраняет учётные данные в хранилище
Главные актёры: Пользователь
Второстепенные актёры: Нет.
Предусловия: 1. Программа запущена

Продолжение таблицы 2

2. Вход авторизован
Основной поток:
1. Пользователь нажимает кнопку сохранить
2. Проверяется мастер-пароль
3. Учётные данные шифруются
Учётные данные сохраняются в хранилище
Постусловия:
Нет.
Альтернативные потоки:
Нет.

Диаграмма вариантов использования и описание прецедентов позволяют вносить правки в будущую программу на этапе анализа требований.

1.3.2 Нефункциональные требования

Список нефункциональных требований:

- программа должна запускаться на ОС Android 6.0+ и iOS 13+;
- программа должна использовать локальное хранилище для доступа к данным;
- время реакции системы на действия пользователя должно составлять не более 100 миллисекунд;
- программа должна занимать не более 256 Мбайт на жестком диске;
- запущенная программа должна использовать не более 32 Мбайт оперативной памяти.

На данном этапе были разработаны функциональные и нефункциональные требования к системе [8].

2 Проектирование системы управления учётными данными

2.1 Проектирование интерфейса управления учётными данными

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

Пользовательский интерфейс должен выполнять следующие требования:

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

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

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

Пользователь должен понимать, как работает программа лишь взглянув на интерфейс. Таким образом интерфейс должен одновременно выполнять роль инструкции. Чтобы достичь этого, необходимо использовать узнаваемые иконки и пиктограммы. Это поможет пользователю быстрее ориентироваться в интерфейсе. Иконки должны быть однозначными и сильно отличаться друг от друга. У пользователя не должно возникать вопроса «А что делает эта кнопка?» Также у хорошего интерфейса должно присутствовать так называемое «пустое состояние». Это момент самого начала работы, когда пользователь еще ничего не сделал и интерфейс в это время должен подсказывать, с чего ему лучше начать.

Интерфейс должен акцентировать внимание на самом важном. Это можно сделать многими способами. Привлечение внимания цветом, пожалуй,

один из самых эффективных. На экране должно присутствовать ограниченное количество элементов. Иначе большое нагромождение вызовет усталость. Заполнение форм можно разделить на несколько страниц.

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

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

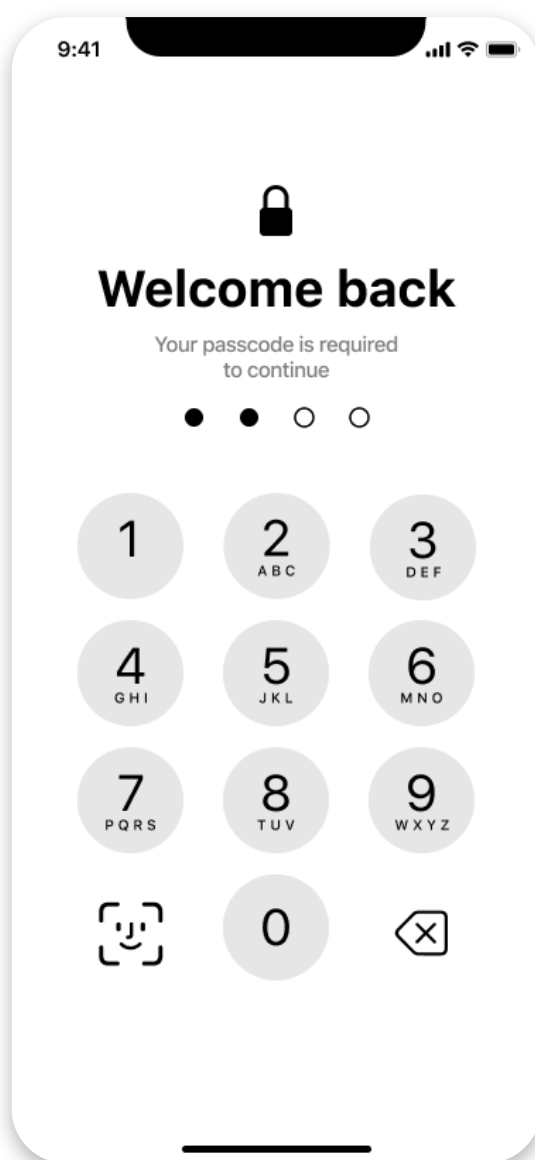


Рисунок 6 – Экран авторизации

На рисунке 6 представлен экран авторизации. На нём имеется клавиатура для ввода ПИН-кода. Точки обозначают длину и введенные цифры. Предоставляется возможность биометрической авторизации по нажатию на соответствующую кнопку (в данном случае на иконку FaceID).

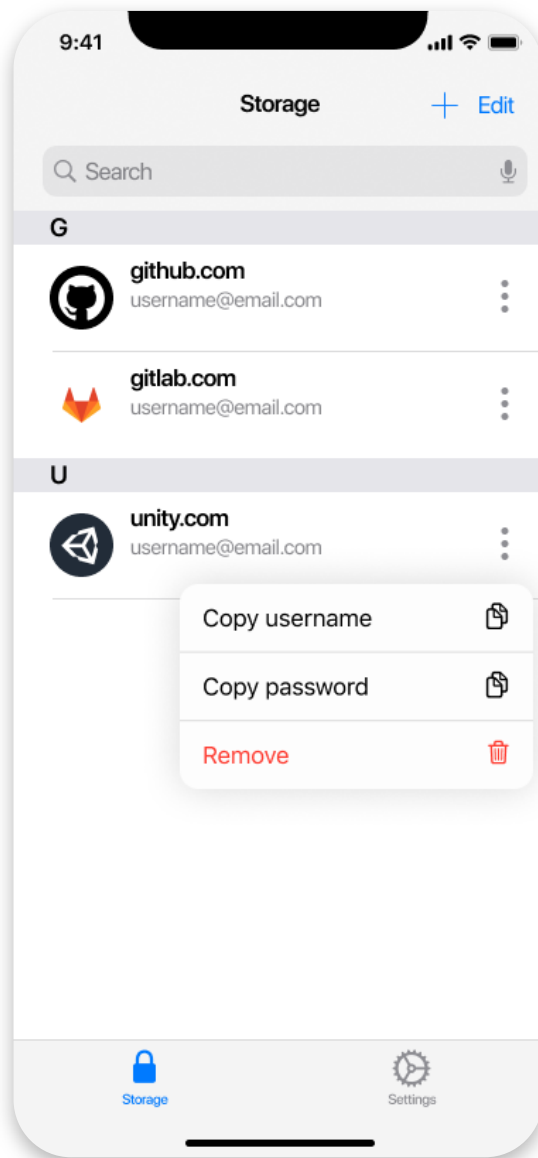


Рисунок 7 – Список учётных записей

На рисунке 7 показан список учётных записей. Это окно открывается после авторизации или при клике на раздел «Storage» в самой нижней панели. По нажатию на «+» можно добавить новую учётную запись. Кнопка «Edit» позволяет выбрать несколько учётных записей и проводить над ними совместные операции. Поле «Search» позволяет найти конкретную учётную запись по идентификатору или имени пользователя. Долгое нажатие по

элементу списка или тап по трём точкам вызывает контекстное меню, где пользователь может скопировать имя пользователя или пароль, а также удалить учётную запись. Клик по элементу списка открывает меню настроек учётной записи, которое показано на рисунке 8.



Рисунок 8 – Настройки учётной записи

В меню настроек учётной записи представлены главные поля каждого элемента. По клику на URL можно перейти на сайт. Имя пользователя и пароль можно скопировать, нажав соответствующую кнопку. Также пароль можно отобразить, нажав на иконку глаза. Чтобы удалить учётную запись, достаточно кликнуть на кнопку «Remove». Чтобы непосредственно изменить поля,

необходимо включить режим редактирования через кнопку «Edit» в верхнем углу.

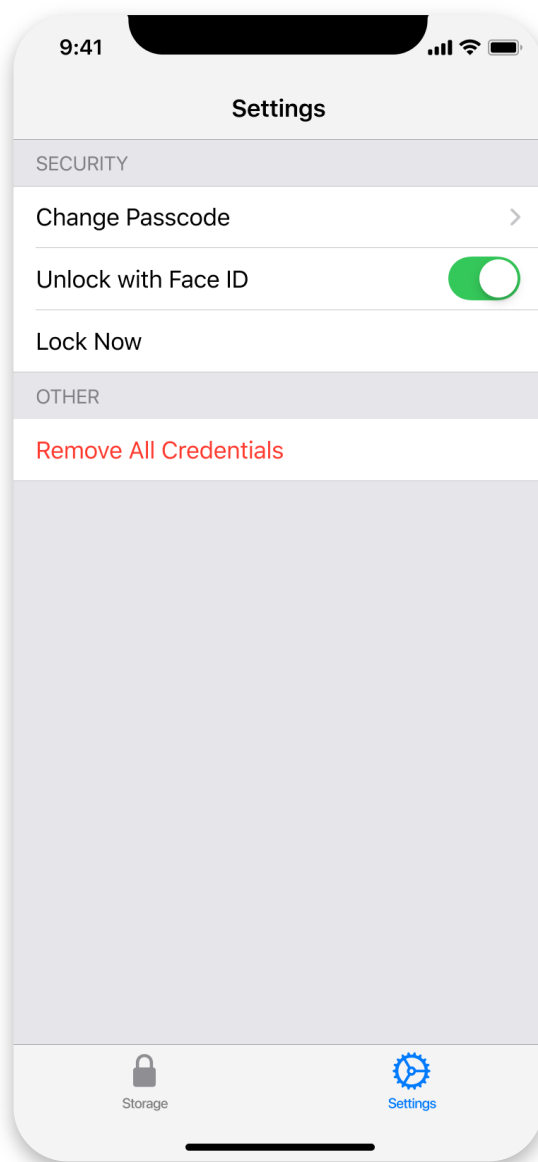


Рисунок 9 – Настройки программы

В меню настроек приложения, которое показано на рисунке 9, пользователь может изменить ПИН-код, включить или выключить биометрическую авторизацию, заблокировать хранилище и удалить все учётные записи с устройства.

2.2 Структура программы управления учётными данными

Для проектирования программы администрирования учётных записей были созданы независимые модули, отвечающие за выполнения определённых задач.

Такой подход облегчает модификацию и сопровождение программного обеспечения, а также необходим для кроссплатформенной работы. Диаграмма компонентов представлена на рисунке 10.

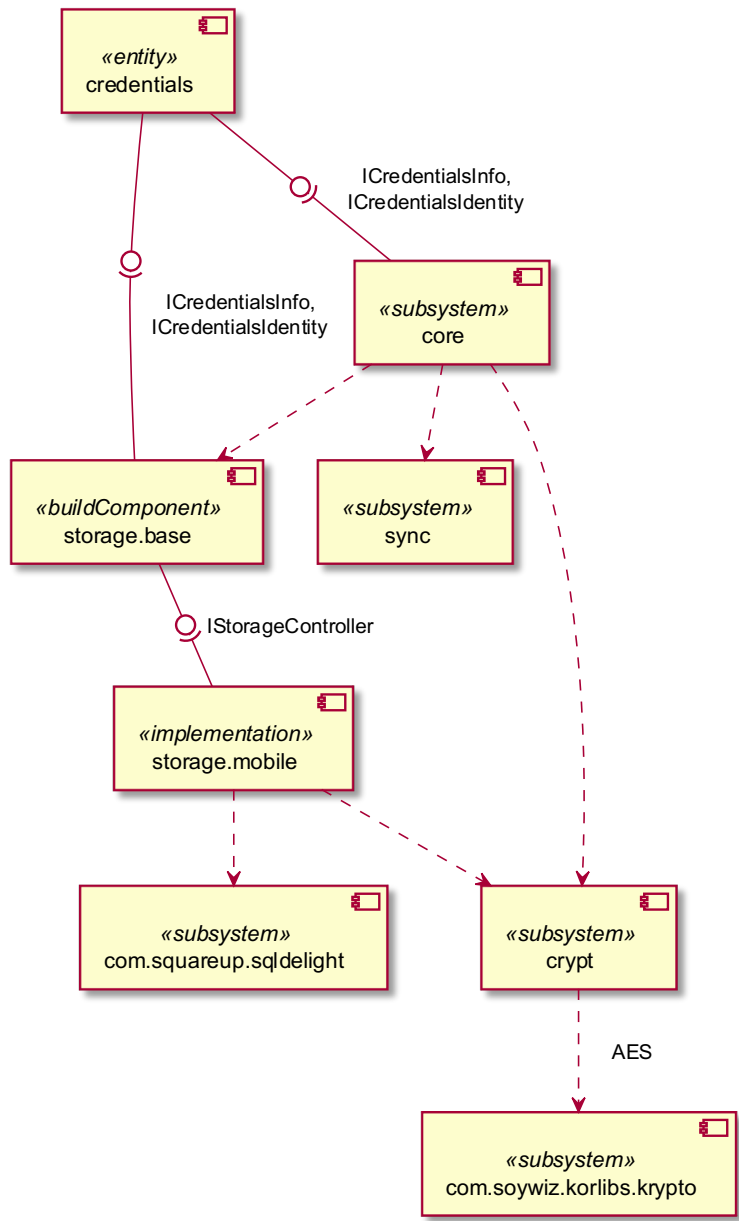


Рисунок 10 – Диаграмма компонентов

Список модулей и их назначение представлены в таблице 3.

Таблица 3 – Модули и их назначение

Название модуля	Назначение
crypt	Содержит методы, отвечающие за шифрование информации и предоставляет методы криптографии.
credentials	Содержит реализацию объектов учётных записей

Продолжение таблицы 3

Название модуля	Назначение
storage.base	Содержит абстракцию для работы с локальной базой данных
storage.mobile	Содержит реализацию драйвера для работы с базой данных для мобильных устройств
core	Предоставляет высокоуровневый интерфейс для работы с учётными данными

В модуле `crypt` содержатся интерфейсы `ICrypt`, `IHasher`. Классы `AESCryptBase`, `AESCryptISO`, `AESCryptZero` реализуют `ICrypt` и предоставляют разные модификации алгоритма шифрования AES. Классы `CRC32` и `SHA256` являются потомками абстрактного `IHasher` и реализуют алгоритмы хеширования. Используется сторонняя библиотека «com.soywiz.korlibs.krypto» в качестве реализации алгоритмов шифрования. На данный момент это единственная стабильная кроссплатформенная библиотека, доступная в Kotlin MPP.

В модуле `credentials` содержатся интерфейсы `ICredentialsIdentity` и `ICredentialsInfo`. `ICredentialsIdentity` представляет собой учётную сущность, которая содержит в себе ограниченное количество информации. Реализован класс `CredentialsIdentity`. `ICredentialsInfo` представляет полные учётные данные и реализован классом `CredentialsInfo`.

В модуле `storage.base` содержится интерфейс `IStorageController`. Она имеет свойство `encryptionKey` – ключ шифрования. В этой абстракции также содержатся методы для получения всех учётных сущностей, учётной записи по учётной сущности, сохранению, удалению, модификации учётных сущностей и метод для очистки базы данных.

Класс `MobileStorageController` в модуле `storage.mobile` является уже конкретной реализацией интерфейса `IStorageController` для мобильных устройств. Для облегчения коммуникации с локальным хранилищем на устройстве используется сторонняя библиотека «com.squareup.sqldelight». Других кроссплатформенных аналогов для Kotlin у неё нет.

Диаграмма классов представлена в приложении Б.

Для хранения учётных записей используется локальная база данных. Она необходима для более быстрого доступа и возможности использования приложения оффлайн. Также локальная база данных является одним из требований гайдлайна Apple [1].

Логическая схема представлена на рисунке 11.

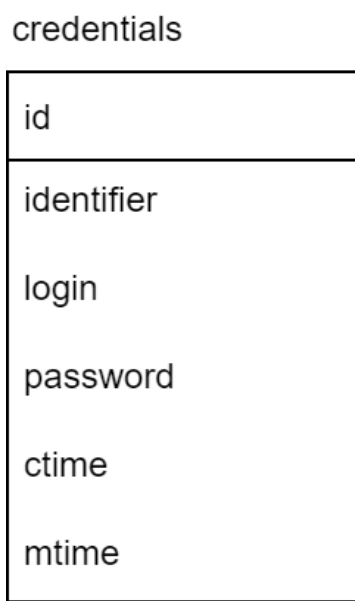


Рисунок 11 – Логическая схема

Схема состоит из одной таблицы `credentials`, которая хранит учётные записи. Для хранения `url` интернет-ресурса используется поле `identifier`, поля `login` и `password` хранят соответственно имя пользователя и пароль, а в полях `ctime` и `mtime` записаны дата создания и дата модификации. Поле `id` является первичным ключом.

2.3 Технологический стек

Для разработки приложения были использованы следующие технологии:

- объектно-ориентированный подход;
- object-relational mapping (ORM);
- кроссплатформенная разработка.

Объектно-ориентированный подход является ключевым аспектом в реализации модульной архитектуры, ORM необходим для простого и безопасного доступа к базе данных [9]. Кроссплатформенная разработка хорошо решает задачу удешевления и ускорения процесса разработки [10].

Исходя из этого был использован следующий стек технологий:

- Kotlin MPP;
- SQLite;
- SQLDelight;
- Git.

Ядро программы для администрирования учётных записей пользователей было разработано с помощью Kotlin MPP. Это позволило держать единую кодовую базу для всех платформ.

Так как Kotlin MPP не поддерживает создание кроссплатформенных интерфейсов, то для их разработки допустимо использовать Flutter.

В качестве локальной системы управления базой данных (СУБД) использовался SQLite. В качестве ORM выбран фреймворк, специально разработанный для SQLite и Kotlin – SQLDelight.

Таким образом программа реализована на языках Kotlin и Dart. Макеты интерфейса проектировались с помощью Figma. В качестве системы контроля версий был использован Git.

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

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

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

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

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

ORM облегчает работу с СУБД и предоставляет безопасное по типам обращение [9].

3 Реализация ядра управления учётными данными

3.1 Программная реализация

В данной главе описана реализация программной модуля локального хранилища для мобильных платформ. Программный код представлен в приложении В.

Фрагменты кода основных функций локального хранилища и их описание приведены далее.

Исходный код функции `getIdentities` приведён в листинге 1.

Листинг 1 – Функция `getIdentities`

```
override fun getIdentities(): StorageReadResult<List<ICredentialsIdentity>> {
    if (encryptionKey == null) {
        return StorageReadResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

    if (!isAccessGranted()) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not correct"
        )
    }

    val list = storage.credentialsQueries.selectAllIdentities { id, identifier, login
->
        decryptCredentialsIdentity(CredentialsIdentity(id, identifier, login))
    }.executeAsList()

    return StorageReadResult(list)
}
```

Функция `getIdentities` отвечает за вывод всех учётных сущностей, хранящихся в базе данных. В качестве результата она возвращает объект класса `StorageReadResult`, с установленным флагом `isSuccess` и значением в `value`, если операция прошла успешно. Иначе будет присвоена переменная `denyReason`, где хранится причина ошибки и `denyMsg` с сообщением об ошибке. Диаграмма деятельности для функции `getIdentities` представлена на рисунке 12.



Рисунок 12 – Диаграмма деятельности getIdentities

Перед запросом в хранилище производится проверка на наличие и валидность ключа шифрования. Если все проверки выполнены успешно, то отправляется запрос в базу данных, данные дешифруются и возвращается результат, содержащий список всех учётных сущностей. Иначе возвращается результат с ошибкой ACCESS_DENIED.

Исходный код функции `getInfo` приведён в листинге 2.

Листинг 2 – Функция `getInfo`

```
override fun getInfo(identity: ICredentialsIdentity):
StorageReadResult<ICredentialsInfo> {
    if (encryptionKey == null) {
        return StorageReadResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

    if (!isAccessGranted()) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not correct"
        )
    }

    if (identity.id == 0L) {
        return StorageReadResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity not saved"
        )
    }

    val value =
        doGetInfo(identity)
        ?: return StorageReadResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity does not exist"
        )

    return StorageReadResult(decryptCredentialsInfo(value))
}
```

Функция `getInfo` отвечает за чтение одной учётной записи. В качестве результата она возвращает объект класса `StorageReadResult`, с установленным флагом `isSuccess` и значением в `value`, если операция прошла успешно.

Перед запросом в хранилище производится проверка на наличие и валидность ключа шифрования. Если эти проверки проходят неуспешно, возвращается результат с ошибкой `ACCESS_DENIED`. Далее проверяется корректность введённой учётной сущности. При неудаче возвращается ошибка `NOT_FOUND`. Производится запрос в базу данных для чтения найденной учётной записи. Если запись не найдена, возвращается ошибка `NOT_FOUND`. Иначе найденная запись дешифруется, и возвращается результат, содержащий объект с интерфейсом `ICredentialsInfo`.

Диаграмма деятельности для функции `getInfo` представлена на рисунке 13.



Рисунок 13 – Диаграмма деятельности `getInfo`

Исходный код функции `saveInfo` приведён в листинге 3.

Листинг 3 – Функция `saveInfo`

```

override fun saveInfo(info: ICredentialsInfo): StorageAccessResult {
    if (encryptionKey == null) {
        return StorageAccessResult(

```

```

        StorageAccessDenyReason.ACCESS_DENIED,
        "Key is not set"
    )
}

if (!isAccessGranted()) {
    return StorageAccessResult(
        StorageAccessDenyReason.ACCESS_DENIED,
        "Key is not correct"
    )
}

if (doGetInfo(info) != null) {
    return StorageAccessResult(
        StorageAccessDenyReason.CONFLICT,
        "Credentials info with id ${info.id} already exists"
    )
}

val eInfo = encryptCredentialsInfo(info)
val savedInstance : ICredentialsInfo? =
storage.credentialsQueries.transactionWithResult {
    storage.credentialsQueries.insertNewCredentials(eInfo.identifier,
eInfo.login, eInfo.password)
    val newID = storage.credentialsQueries.selectLastInsertedId().executeAsOne()
    doGetInfo(CredentialsIdentity(newID, eInfo.identifier, eInfo.login))
}

if (savedInstance != null) {
    info.id = savedInstance.id
    info.ctime = savedInstance.ctime
    info.mtime = savedInstance.mtime
}

return StorageAccessResult()
}

```

Функция `saveInfo` отвечает за сохранение новой учётной записи. В качестве результата она возвращает объект класса `StorageAccessResult`, с установленным флагом `isSuccess`, если операция прошла успешно. Иначе будет присвоена переменная `denyReason`, где хранится причина ошибки и `denyMsg` с сообщением об ошибке.

Проводится проверка на существование ключа шифрования и на его валидность. Далее проверяется корректность введённой учётной записи. Если все предыдущие этапы выполнены успешно, программа посылает запрос в базу данных, чтобы проверить, есть ли уже такие данные. При удачном поиске возвращается результат с ошибкой `CONFLICT`. Иначе данные шифруются и сохраняются в хранилище. Во время сохранения объекту в оперативной памяти присваивается `id` и дата создания.

Диаграмма деятельности для функции saveInfo представлена на рисунке 14.

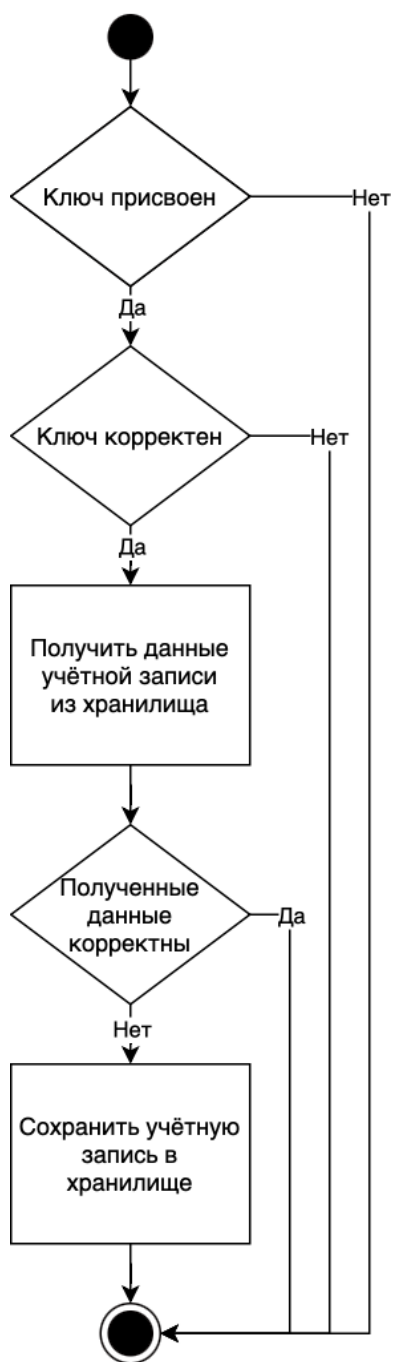


Рисунок 14 – Диаграмма деятельности saveInfo

Исходный код функции updateInfo приведён в листинге 4.

Листинг 4 – Функция updateInfo

```

override fun updateInfo(info: ICredentialsInfo): StorageAccessResult {
    if (encryptionKey == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }
}
  
```

```

if (!isAccessGranted()) {
    return StorageAccessResult(
        StorageAccessDenyReason.ACCESS_DENIED,
        "Key is not correct"
    )
}

if (info.id == 0L) {
    return StorageAccessResult(
        StorageAccessDenyReason.NOT_FOUND,
        "Identity not saved"
    )
}

if (doGetInfo(info) == null) {
    return StorageAccessResult(
        StorageAccessDenyReason.NOT_FOUND,
        "Credentials info with id ${info.id} already exists"
    )
}

val eInfo = encryptCredentialsInfo(info)
storage.credentialsQueries.transaction {
    storage.credentialsQueries.updateCredentials(
        eInfo.identifier,
        eInfo.login,
        eInfo.password,
        eInfo.ctime,
        eInfo.id
    )
    info.mtime = doGetInfo(eInfo)?.mtime ?: 0
}

return StorageAccessResult()
}

```

Функция `updateInfo` отвечает за обновление имеющейся учётной записи. В качестве результата она возвращает объект класса `StorageAccessResult`

Проводится проверка на существование ключа шифрования и на его валидность. Далее проверяется корректность введённой учётной записи. Далее проверяется корректность введённой учётной записи. При неудаче возвращается ошибка `NOT_FOUND`. Производится запрос в базу данных для чтения найденной учётной записи. Если запись не найдена, возвращается ошибка `NOT_FOUND`. Данные шифруются и сохраняются в хранилище. Во время сохранения у объекта в оперативной памяти обновляется дата изменения.

Диаграмма деятельности для функции `deleteInfo` представлена на рисунке 15.

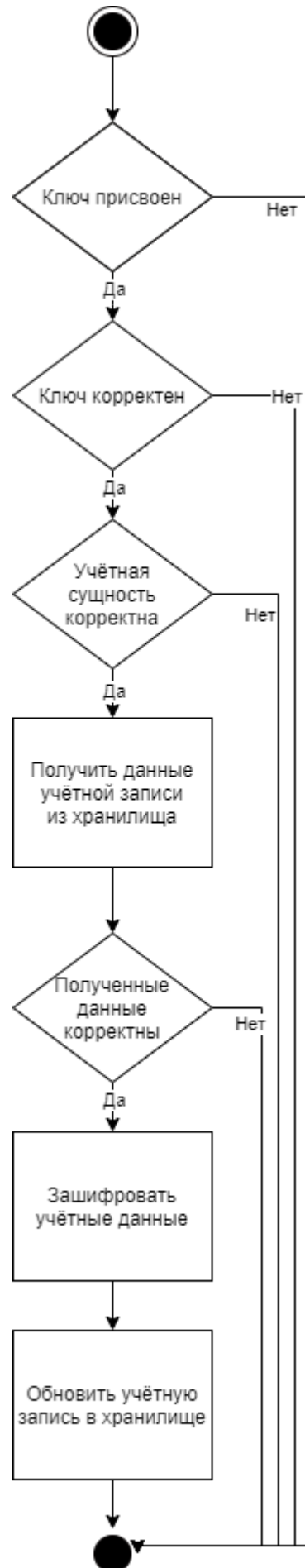


Рисунок 15 – Диаграмма деятельности updateInfo

Исходный код функции deleteInfo приведён в листинге 5.

Листинг 5 – Функция deleteInfo

```
override fun deleteInfo(info: ICredentialsInfo): StorageAccessResult {
    if (encryptionKey == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

    if (!isAccessGranted()) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not correct"
        )
    }

    if (info.id == 0L) {
        return StorageAccessResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity not saved"
        )
    }

    if (doGetInfo(info) == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Credentials info with id ${info.id} does not exist"
        )
    }

    storage.credentialsQueries.deleteCredentials(info.id)

    return StorageAccessResult()
}
```

Функция deleteInfo отвечает за удаление одной учётной записи. В качестве результата она возвращает объект класса StorageAccessResult.

Проводится проверка на существование ключа шифрования и на его валидность. Далее проверяется корректность введенной учётной записи. Далее проверяется корректность введенной учётной записи. При неудаче возвращается ошибка NOT_FOUND. Производится запрос в базу данных для чтения найденной учётной записи. Если запись не найдена, возвращается ошибка NOT_FOUND. Если все условия успешно выполняются, то производится запрос в базу данных для удаления найденной учётной записи.

Диаграмма деятельности для функции deleteInfo представлена на рисунке 16.



Рисунок 16 – Диаграмма деятельности deleteInfo

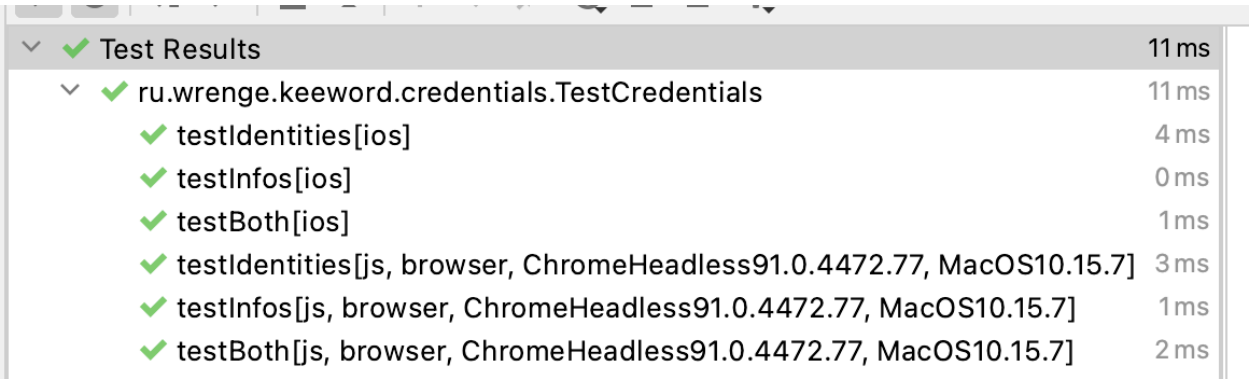
В данной главе описана реализация ядра управления учётными данными. Программный код представлен в приложении В.

3.2 Тестирование ядра управления учётными данными

Так как разработанная программа представляет собой библиотеку, то единственным вариантом тестирования является модульное.

Были написаны модульные тесты для всех компонентов программы, которые проверяют работоспособность на всех платформах. Код модульных тестов приведён в приложении В.

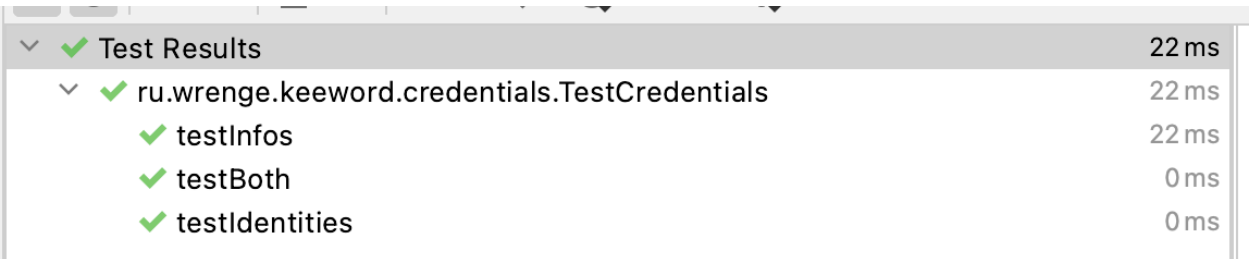
В компоненте `credentials` было протестирована функция равенства между разными сущностями. Была выявлена и исправлена ошибка при проверке равенства объекта класса `ClassicCredentialsInfo` и объекта класса `CredentialsIdentity`. При сравнении этих объектов с одинаковыми значениями в полях результат менялся в зависимости от постановки объектов. Ошибка была в функции `equals` класса `ClassicCredentialsInfo`. Добавлена дополнительная проверка на то, имеет ли сравниваемый объект интерфейс `ICredentialsInfo`, и если это так, то вызывается функция этого объекта. Результаты повторного тестирования представлены на рисунках 17 и 18.



The screenshot shows a test results window with a tree view. The root node is 'Test Results' with a green checkmark and a duration of 11 ms. It has a sub-node 'ru.wrenge.keeword.credentials.TestCredentials' also with a green checkmark and 11 ms. Under this sub-node are seven test cases, all with green checkmarks: 'testIdentities[ios]' (4 ms), 'testInfos[ios]' (0 ms), 'testBoth[ios]' (1 ms), 'testIdentities[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]' (3 ms), 'testInfos[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]' (1 ms), and 'testBoth[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]' (2 ms).

Test Results	Duration
Test Results	11 ms
ru.wrenge.keeword.credentials.TestCredentials	11 ms
testIdentities[ios]	4 ms
testInfos[ios]	0 ms
testBoth[ios]	1 ms
testIdentities[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]	3 ms
testInfos[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]	1 ms
testBoth[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15.7]	2 ms

Рисунок 17 – Результат тестирования компонента `credentials` на платформах iOS и JavaScript

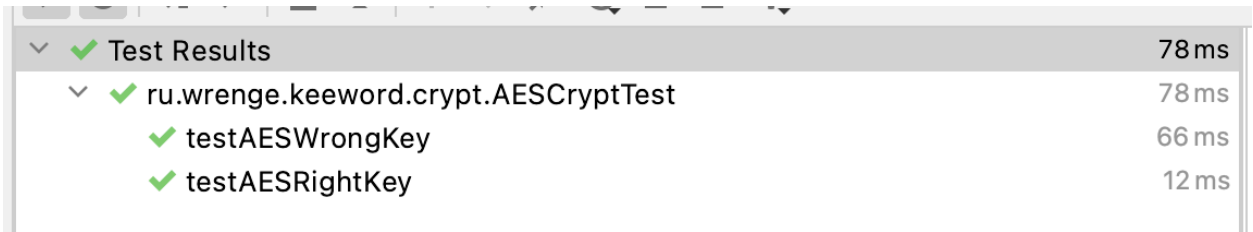


The screenshot shows a test results window with a tree view. The root node is 'Test Results' with a green checkmark and a duration of 22 ms. It has a sub-node 'ru.wrenge.keeword.credentials.TestCredentials' also with a green checkmark and 22 ms. Under this sub-node are four test cases, all with green checkmarks: 'testInfos' (22 ms), 'testBoth' (0 ms), and 'testIdentities' (0 ms). The 'testIdentities' node is not expanded in the image.

Test Results	Duration
Test Results	22 ms
ru.wrenge.keeword.credentials.TestCredentials	22 ms
testInfos	22 ms
testBoth	0 ms
testIdentities	0 ms

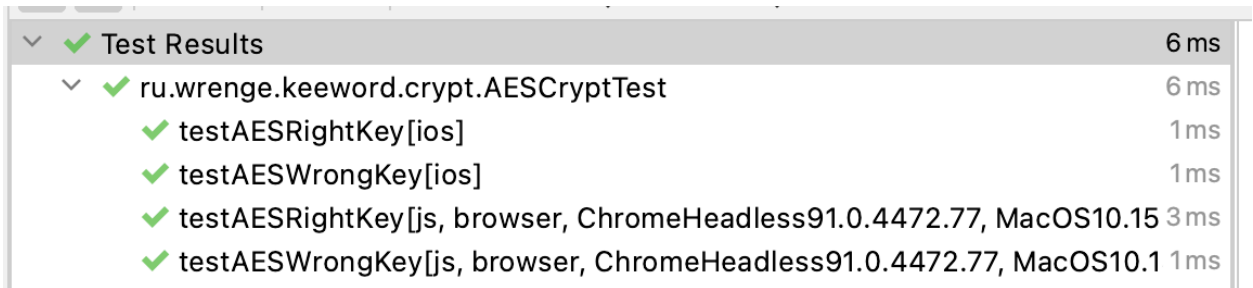
Рисунок 18 – Результат тестирования компонента `credentials` на платформе Android

В компоненте `crypt` были протестированы функции шифрования и дешифрования с правильным и некорректным ключами. Тестирование было успешно пройдено без ошибок. Результаты тестирования представлены на рисунках 19 и 20.



✓ Test Results	78 ms
✓ ru.wrengе.кеeword.crypt.AESCryptTest	78 ms
✓ testAESWrongKey	66 ms
✓ testAESRightKey	12 ms

Рисунок 19 – Результат тестирования компонента `crypt` на платформе Android



✓ Test Results	6 ms
✓ ru.wrengе.кеeword.crypt.AESCryptTest	6 ms
✓ testAESRightKey[ios]	1ms
✓ testAESWrongKey[ios]	1ms
✓ testAESRightKey[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15]	3 ms
✓ testAESWrongKey[js, browser, ChromeHeadless91.0.4472.77, MacOS10.15]	1 ms

Рисунок 20 – Результат тестирования компонента `crypt` на платформах iOS и JavaScript

В компоненте `storage-base` не проводится тестирования, так как он не содержит никакой логики и просто предоставляет интерфейс хранилища.

В компоненте `crypt` были протестированы все реализованные функции интерфейса `IStorageController`. Была выявлена и исправлена ошибка, когда при сохранении новой учётной записи и запросе её `id`, база данных всегда возвращала ноль. Это связано с особенностью реализации фреймворка `SQLDelight`. Чтобы исправить эту ошибку, все запросы к базе данных были по возможности сгруппированы в единые транзакции. Результаты повторного тестирования представлены на рисунках 21 и 22.

✓ Test Results	5 sec 426 ms
✓ ru.wrenge.keeword.storage.mobile.DBControllerTest	5 sec 426 ms
✓ testAccessNoKey	4 sec 196 ms
✓ testClearInfos	782 ms
✓ testSaveInfo	83 ms
✓ testUpdateInfo	95 ms
✓ testGetInfo	79 ms
✓ testDeleteInfo	90 ms
✓ testDeleteDB	101 ms

Рисунок 21 – Результат тестирования компонента storage.mobile на платформе Android

✓ Test Results	150 ms
✓ ru.wrenge.keeword.storage.mobile.DBControllerTest	150 ms
✓ testAccessNoKey[ios]	22 ms
✓ testGetInfo[ios]	40 ms
✓ testSaveInfo[ios]	14 ms
✓ testUpdateInfo[ios]	16 ms
✓ testDeleteInfo[ios]	15 ms
✓ testClearInfos[ios]	18 ms
✓ testDeleteDB[ios]	25 ms

Рисунок 22 – Результат тестирования компонента storage.mobile на платформах iOS

Таким образом, была осуществлена проверка правильности работы разработанного программного обеспечения.

3.3 Расчёт метрик кода системы управления учётными данными

Помимо проверки правильности работы программы необходимо проверить качество кода для того, чтобы убедиться в правильности проектирования архитектуры. Это позволит в дальнейшем дорабатывать и расширять программный продукт, не прибегая к рефакторингу кода.

Метрики кода позволяют получить некоторые численные значения каких-либо свойств программного обеспечения. Метрики кода зачастую не показывают, насколько хорошо написан программный код. Любые метрики несовершенны и не учитывают ситуации, когда нестандартное архитектурное

решение работает лучше остальных решений с более хорошими значениями метрик.

На рисунке 23 представлен объем репозитория разработанной программы.

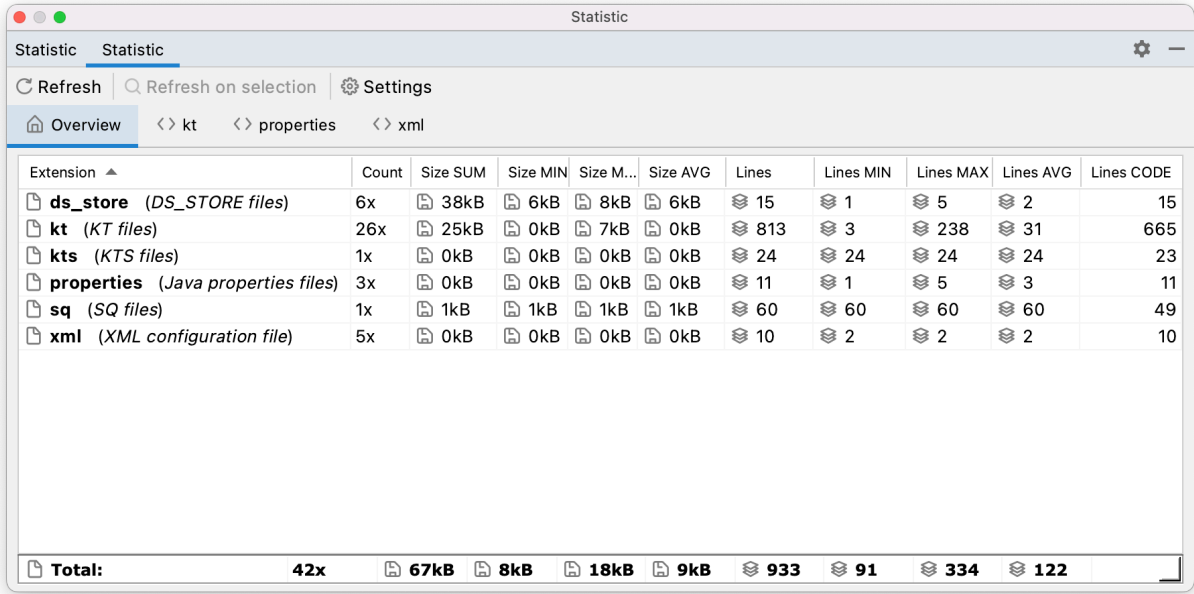


Рисунок 23 – Объем репозитория

Репозиторий программного обеспечения состоит из исходного кода на языке Kotlin (.kt), исходного кода окружения (.kts), разработанной схемы базы данных на языке SQL в диалекте SQLDelight (.sq). Остальные файлы являются производным от операционной системы и встроенной среды разработки.

Всего к репозиторию относится 36 файлов (файлы ds_store не учитываются) общим объёмом в 27 Кб. Среднее количество строк в файле с исходным кодом – 31.

Кодовая база получилась достаточно компактной – всего 813 строк. Из них в 665 записан код. Остальные не отвечают за работоспособность программы. Это пустые строки и комментарии, которые нужны для лучшей читаемости кода.

На рисунке 24 представлен объем исходного кода разработанной программы.

Source File	Total Lines	Source Cod...	Source Code Li...	Comment Lines	Comment Lines ...	Blank Lines	Blank Lines [%]
BaseStorageController.kt	238	200	84%	0	0%	38	16%
DBControllerTestBase.kt	226	175	77%	0	0%	51	23%
TestCredentials.kt	51	44	86%	0	0%	7	14%
ClassicCredentialsInfo.kt	37	32	86%	0	0%	5	14%
AESCryptBase.kt	38	31	82%	0	0%	7	18%
CredentialsIdentity.kt	26	22	85%	0	0%	4	15%
AESCryptTest.kt	25	20	80%	0	0%	5	20%
DBControllerTest.kt	23	19	83%	0	0%	4	17%
IStorageController.kt	17	15	88%	0	0%	2	12%
DBControllerTest.kt	14	11	79%	0	0%	3	21%
StorageReadResult.kt	12	10	83%	0	0%	2	17%
StorageController.kt	11	9	82%	0	0%	2	18%
StorageController.kt	10	8	80%	0	0%	2	20%
StorageAccessDenyReason.kt	9	8	89%	0	0%	1	11%
SHA256.kt	9	7	78%	0	0%	2	22%
StorageAccessResult.kt	8	7	88%	0	0%	1	12%
CRC32.kt	7	6	86%	0	0%	1	14%
CredentialsIdentity.kt	7	6	86%	0	0%	1	14%
Total:	813	665	82%	0	0%	148	18%

Рисунок 24 – Объем исходного кода

Таким образом, были рассчитаны метрики кода разработанного программного обеспечения.

3.4 Планирование разработки и оценка бюджета

Разработку программы можно разделить на следующие части:

- разработка пользовательского интерфейса;
- разработка ядра;
- привязка интерфейса и ядра.

Таким образом разработка программного обеспечения администрирования учётных записей пользователей может осуществляется следующими специалистами:

- разработчик пользовательских интерфейсов;
- разработчик клиентского приложения.

В задачи верстальщика пользовательских интерфейсов входит дизайн макета пользовательского интерфейса, вёрстка и программирование точек входа и выхода для привязки ядра.

В задачи разработчика клиентского приложения входит проектирование архитектуры ядра, схемы локальной базы данных будущего программного продукта, а также их реализация, отладка и сопровождение.

Для разработки плана создания программного обеспечения для администрирования учётных записей пользователя было решено использовать средство управления проектами Microsoft Project Pro [11].

План разработки с распределением участников разработки по задачам представлен на рисунке 25.

	Режим задачи	Название задачи	Длительность	Начало	Окончание	Предшественники	Названия ресурсов
1	✓	Разработка проекта	34 дней	Чт 27.05.21	Вт 13.07.21		
2	✓	Начало разработки проекта	0 дней	Чт 27.05.21	Чт 27.05.21		
3	✓	Закупка ресурсов	1 день	Чт 27.05.21	Чт 27.05.21		MacBook Air 13"[0];Компьютер[0];IntelliJ IDEA[0];J
4	✓	Разработка пользовательского интерфейса	6 дней	Чт 27.05.21	Чт 03.06.21		
5	✓	Дизайн макета	2 дней	Чт 27.05.21	Пт 28.05.21		Верстальщик;Электроэнергия[200,00 Р]
6	✓	Верстка интерфейса	3 дней	Пт 28.05.21	Ср 02.06.21	5	Верстальщик;Электроэнергия[300,00 Р]
7	✓	Создание точек входа и выхода	1 день	Ср 02.06.21	Чт 03.06.21	6	Верстальщик;Электроэнергия[100,00 Р]
8	✓	Разработка ядра	16 дней	Чт 27.05.21	Чт 17.06.21		
9	✓	Проектирование архитектуры	0,5 дней	Чт 27.05.21	Чт 27.05.21		Программист;Электроэнергия[50,00 Р]
10	✓	Проектирование схемы базы данных	0,5 дней	Чт 27.05.21	Чт 27.05.21	9	Программист;Электроэнергия[50,00 Р]
11	✓	Реализация модуля credentials	0,5 дней	Пт 28.05.21	Пт 28.05.21	9	Программист;Электроэнергия[50,00 Р]
12	✓	Отладка модуля credentials	2,5 дней	Пт 28.05.21	Вт 01.06.21	11	Программист;Электроэнергия[250,00 Р]
13	✓	Реализация модуля crypt	1 день	Ср 02.06.21	Ср 02.06.21	9	Программист;Электроэнергия[100,00 Р]
14	✓	Отладка модуля crypt	3 дня	Ср 02.06.21	Пн 07.06.21	13	Программист;Электроэнергия[300,00 Р]
15	✓	Реализация модуля storage-base	0,5 дней	Вт 08.06.21	Вт 08.06.21	9	Программист;Электроэнергия[50,00 Р]
16	✓	Реализация схемы базы данных	1,5 дней	Вт 08.06.21	Ср 09.06.21	10;15	Программист;Электроэнергия[150,00 Р]
17	✓	Реализация модуля storage-mobile	2 дня	Ср 09.06.21	Пт 11.06.21	9;16	Программист;Электроэнергия[200,00 Р]
18	✓	Отладка модуля storage-mobile	3 дня	Пт 11.06.21	Ср 16.06.21	17	Программист;Электроэнергия[300,00 Р]
19	✓	Сборка ядра	1 день	Ср 16.06.21	Чт 17.06.21	12;14;18	Программист;Электроэнергия[500,00 Р]
20	✓	Привязка интерфейса и ядра	18 дней	Чт 17.06.21	Вт 13.07.21		
21	✓	Создание проекта Android	0,5 дней	Чт 17.06.21	Пт 18.06.21	19	Программист;Электроэнергия[50,00 Р]
22	✓	Привязка интерфейсов Android	2,5 дней	Пт 18.06.21	Вт 22.06.21	21;7	Программист;Электроэнергия[250,00 Р]
23	✓	Отладка проекта Android	3 дня	Вт 22.06.21	Пт 25.06.21	22	Программист;Электроэнергия[300,00 Р]
24	✓	Создание проекта iOS	0,5 дней	Вт 29.06.21	Вт 29.06.21	19	Программист;Электроэнергия[50,00 Р]
25	✓	Привязка интерфейсов iOS	2,5 дней	Вт 29.06.21	Чт 01.07.21	24;7	Программист;Электроэнергия[250,00 Р]
26	✓	Отладка проекта iOS	3 дня	Чт 01.07.21	Вт 06.07.21	25	Программист;Электроэнергия[300,00 Р]
27	✓	Создание проекта JavaScript	0,5 дней	Вт 06.07.21	Вт 06.07.21	19	Программист;Электроэнергия[50,00 Р]
28	✓	Привязка интерфейсов JavaScript	2,5 дней	Вт 06.07.21	Чт 08.07.21	27;7	Программист;Электроэнергия[250,00 Р]
29	✓	Отладка проекта JavaScript	3 дня	Чт 08.07.21	Вт 13.07.21	28	Программист;Электроэнергия[300,00 Р]

Рисунок 25 – Календарный план

Как видно из приведённого выше плана, вся разработка, включая составление документации, занимает 34 дня.

По данному графику была составлена диаграмма Ганта, которая представлена в приложении Б.

Как видно из диаграммы Ганта, критический путь составляет 34 календарных дня. Начало разработки назначено на 27 мая 2021 года, окончание – на 13 июня 2021 года.

Себестоимость программного продукта включает в себя следующие затраты:

- заработная плата разработчиков;
- оплата оборудования;
- оплата программного обеспечения;

- различные дополнительные издержки.

Для разработки программы необходимо нанять команду, состоящую из четырех разработчиков:

- разработчик пользовательских интерфейсов;
- разработчик клиентского приложения.

Заработная плата этих специалистов приведена на рисунке 26.

Название ресурса	Тип	Макс. единиц	Стандартная ставка	Начисление	Затраты на исполыз.	Базовый календарь
Программист	Трудовой	100%	55 000,00 Р/мес	Пропорциональное	0,00 Р	Стандартный
Верстальщик	Трудовой	100%	45 000,00 Р/мес	Пропорциональное	0,00 Р	Стандартный

Рисунок 26 – Заработная плата разработчиков

Также необходимо приобрести технику для разработки программы для администрирования учётных записей пользователей и обеспечить разработчиков необходимым программным обеспечением. Необходимые затраты представлены на рисунке 27.

Название ресурса	Тип	Затраты на исполыз.	Начисление
MacBook Air 13"	Материальн	70 000,00 Р	В начале
Компьютер	Материальн	30 000,00 Р	В начале
IntelliJ IDEA	Материальн	36 740,00 Р	В начале
Лицензия Apple Store	Материальн	7 400,00 Р	В начале
Windows 10	Материальн	5 190,00 Р	В начале

Рисунок 27 – Дополнительные затраты

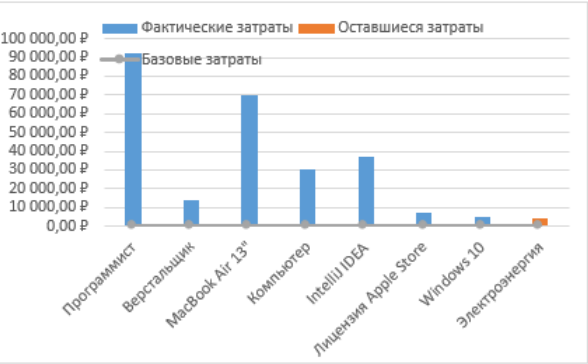
Для разработки требуется компьютер для проектирования и вёрстки пользовательского интерфейса, MacBook Air написания и отладки программы на максимальном диапазоне поддерживаемых устройств, пакет лицензионного программного обеспечения Microsoft и JetBrains.

Необходимо рассчитать окончательный бюджет проекта. Для этого составим отчет в программе Microsoft Project. Он представлен на рисунке 28.

ОБЗОР ЗАТРАТ РЕСУРСОВ

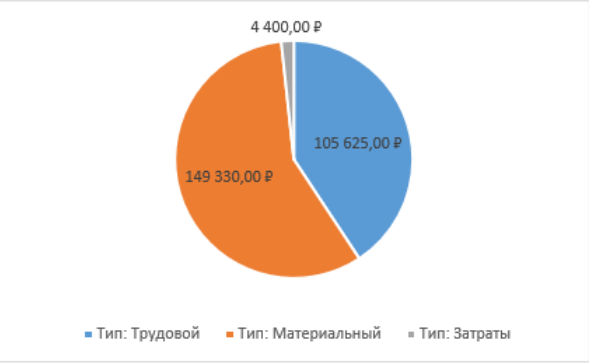
СОСТОЯНИЕ ЗАТРАТ

Состояние затрат для трудовых ресурсов.



РАСПРЕДЕЛЕНИЕ ЗАТРАТ

Распределение затрат между различными типами ресурсов.



СВЕДЕНИЯ О ЗАТРАТАХ

Сведения о затратах для всех трудовых ресурсов.

Название	Затраты	Фактические трудоzатраты	Стандартная ставка
Группа: Нет значения	259 355,00 Р	320 ч	
Программист	92 125,00 Р	272 ч	55 000,00 Р/мес
Верстальщик	13 500,00 Р	48 ч	45 000,00 Р/мес
MacBook Air 13"	70 000,00 Р	1	0,00 Р
Компьютер	30 000,00 Р	1	0,00 Р
IntelliJ IDEA	36 740,00 Р	1	0,00 Р
Лицензия Apple Store	7 400,00 Р	1	0,00 Р
Windows 10	5 190,00 Р	1	0,00 Р
Электроэнергия	4 400,00 Р		

Рисунок 28 – Бюджет разработки проекта

В итоге срок разработки составил 34 дня, бюджет составил 259355 рублей.

Заключение

В рамках выпускной квалификационной работы был осуществлен анализ требований к программному обеспечению администрирования учётных записей пользователей, также были изучены основные принципы их работы и произведен анализ нескольких зарубежных аналогов.

На этапе проектирования была разработана архитектура клиентского ядра менеджера паролей и его локального хранилища, также был спроектирован пользовательский интерфейс. Далее были выбраны технологии разработки: объектно-ориентированное программирование, кроссплатформенная разработка, Kotlin MPP, SQLDelight, язык программирования Kotlin. Был разработан сам интерфейс ядра, в котором был реализован следующий функционал:

- доступ к учётным сущностям;
- сохранение учётных данных;
- обновление учётных данных;
- удаление учётных данных;
- проверка мастер-пароля при доступе к хранилищу;
- очистка хранилища.

Также было проведено тестирование программы, показавшее правильность ее работы, обнаружены и исправлены ошибки, и рассчитаны метрики кода, показавшие, что разработанная программа действительно компактна.

Для организации разработки программного обеспечения администрирования учётных записей пользователей были рассчитаны сроки выполнения работ и подсчитан бюджет.

Список использованных источников

1. Слабые пароли могут вам слишком дорого обойтись // Avast blog [Электронный ресурс] URL: <https://blog.avast.com/ru/slabye-paroli-mogut-vam-slishkom-dorogo-obojtis> (дата обращения: 20.04.2021)
2. Как выбрать менеджер паролей? // Avast blog [Электронный ресурс] URL: <https://blog.avast.com/ru/kak-vybrat-menedzher-parolej> (дата обращения: 20.04.2021)
3. How it works – Dashlane // [Электронный ресурс]. URL: <https://www.dashlane.com/features>. (дата обращения: 20.04.2021)
4. How it works – LastPass // [Электронный ресурс]. URL: <https://www.lastpass.com/how-lastpass-works> (дата обращения: 20.04.2021)
5. LastPass Hacked – Identified Early & Resolved // Blog LastPass [Электронный ресурс]. URL: <http://blog.lastpass.com/2015/06/lastpass-security-notice/> (дата обращения: 20.04.2021)
6. KeePass features [Электронный ресурс] // URL: <https://keepass.info/features.html> (дата обращения: 24.04.2021)
7. MacOS X Security [Электронный ресурс] // URL: https://web.archive.org/web/20110410065128/https://images.apple.com/macosx/security/docs/MacOSX_Security_TB.pdf (дата обращения: 24.04.2021)
8. Implementing AutoFill Credential Provider Extensions // Developer Apple [Электронный ресурс] URL: <https://developer.apple.com/videos/play/wwdc2018/721/> (дата обращения: 20.04.2021)
9. Введение в ORM // Блог веб-разработчика [Электронный ресурс]. URL: <http://internetka.in.ua/orm-intro/> (дата обращения: 20.04.2021)
10. Самсонкин А.С., Такташкин Д.В. Обзор и сравнительный анализ кроссплатформенных средств разработки // сборник трудов VIII Ежегодной всероссийской межвузовской научно-практической конференции «Информационные технологии в науке и образовании. Проблемы и перспективы.» – изд-во ПГУ, 2021

11. Управление проектами средствами Microsoft Project // НОУ ИНТУИТ [Электронный ресурс]. <https://intuit.ru/studies/courses/496/352/info> (дата обращения: 28.05.2021)
12. Unit test – definition // [Электронный ресурс] The art of unit testing. URL: <https://www.artofunittesting.com/definition-of-a-unit-test> (дата обращения: 28.05.2021)
13. Unified Modeling Language [Электронный ресурс] // URL: <https://www.omg.org/spec/UML/2.5.1/PDF> (дата обращения: 04.06.2021)

Приложение А
ГЛОССАРИЙ
(справочное)

Учётные данные – данные пользователя, использующиеся для идентификации и получения доступа к определённым ресурсам. Обычно представляет собой пару имя пользователя (email) и пароль.

Кроссплатформенность – способность программного обеспечения работать с несколькими аппаратными платформами или операционными системами.

Средства кроссплатформенной разработки – набор средств, состоящий из высокоуровневых языков программирования, сред разработки и выполнения, поддерживающих условную компиляцию, компоновку и выполнение кода для различных платформ, обеспечивающих тем самым работу разрабатываемого приложения на нескольких платформах.

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними [13].

Диаграмма компонентов позволяет определить состав программных компонентов, в роли которых может выступать исходный, бинарный и исполняемый код, а также установить зависимости между ними [13].

Компонент – это физическая часть системы, представляющая собой файл с исходным кодом классов, библиотеки, исполняемые модули и т.п. [13].

Модульное тестирование – это автоматизированный вызов фрагмента кода с целью проверки предположения о корректности его поведения [12].

Приложение Б
ДИАГРАММЫ ПРОГРАММЫ
(рекомендуемое)

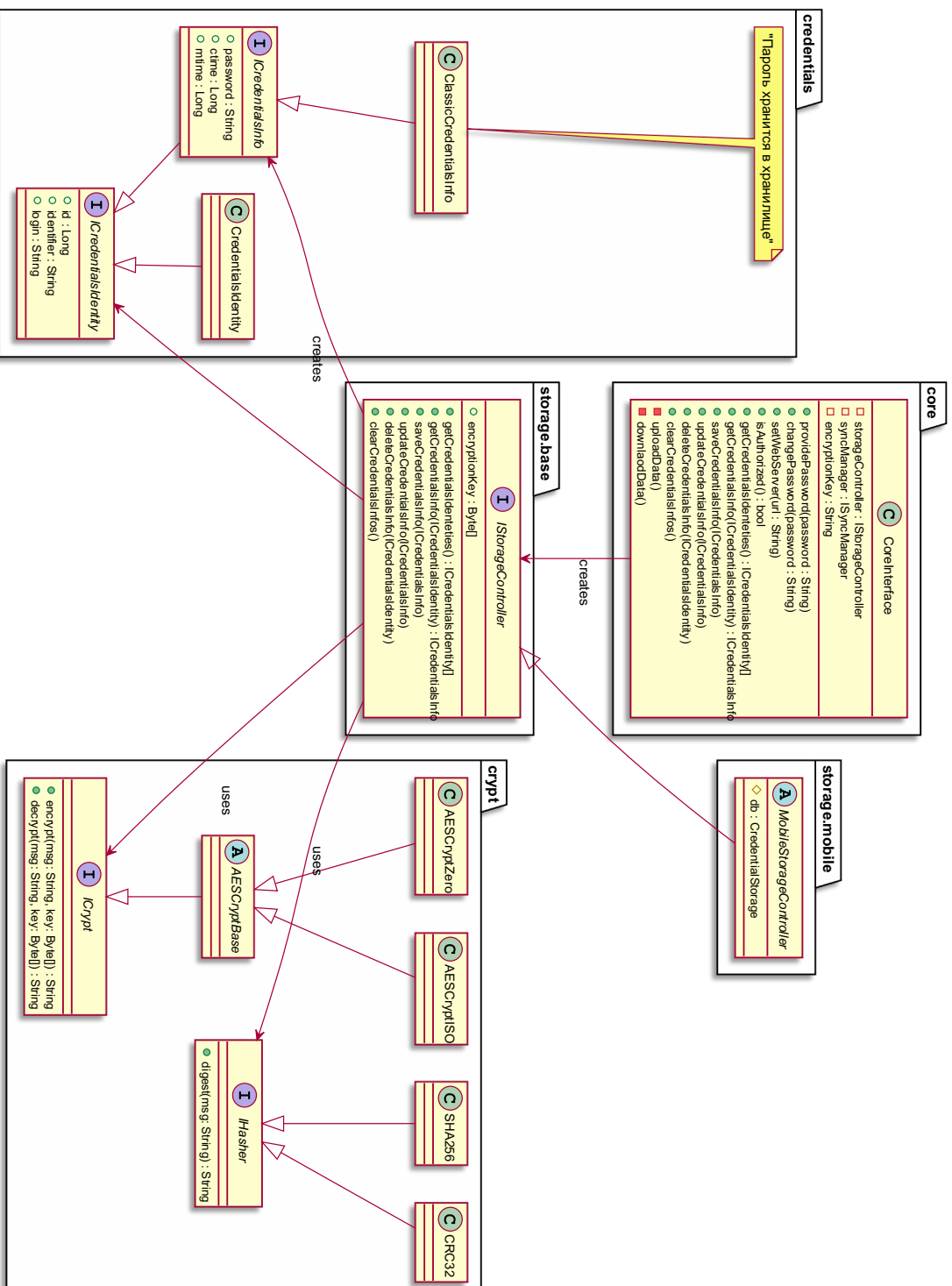


Рисунок Б.1 – Диаграмма классов

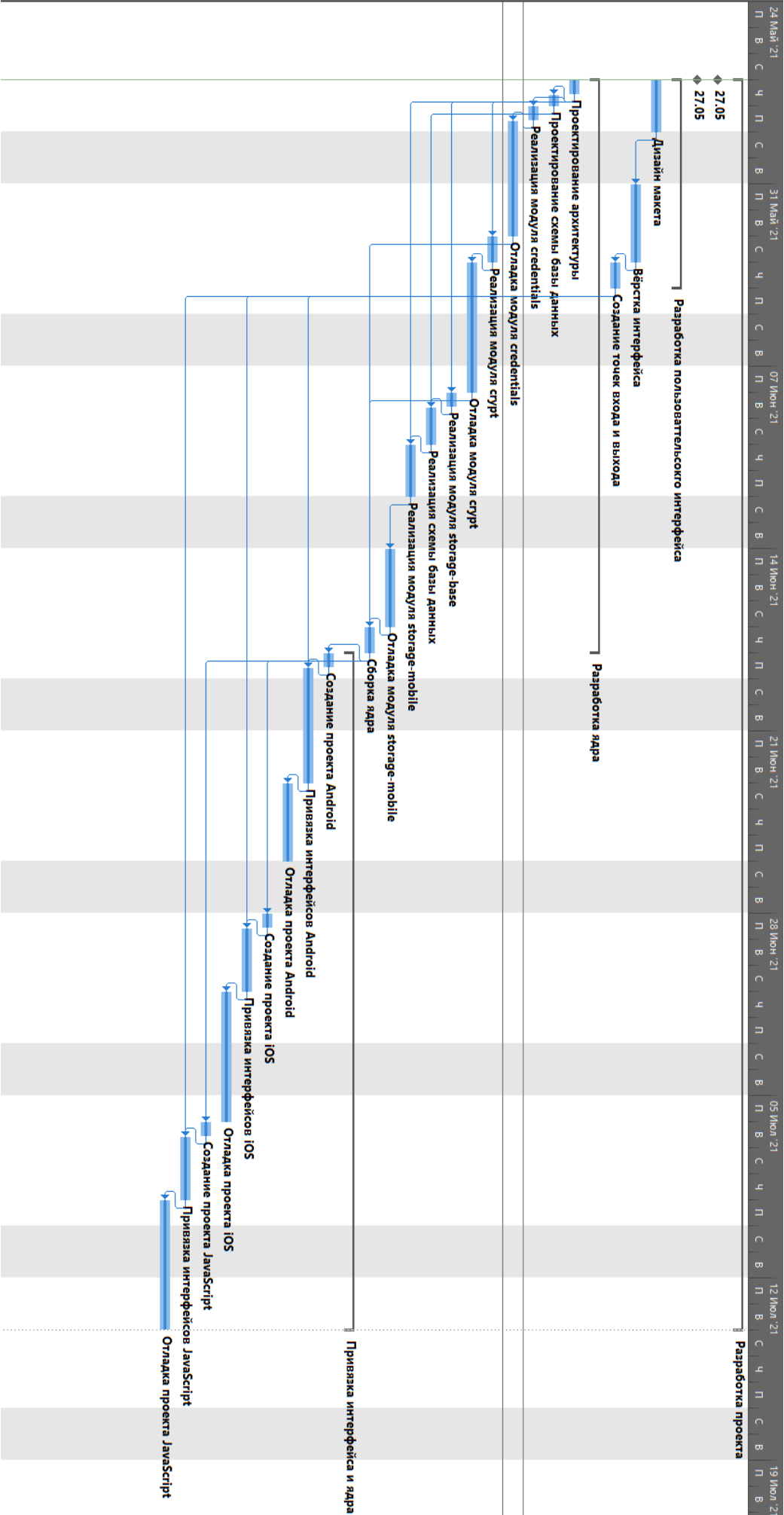


Рисунок Б.2 – Диаграмма Ганта

Приложение В
КОД ПРОГРАММЫ И МОДУЛЬНЫХ ТЕСТОВ
(рекомендуемое)

modules\credentials\src\commonMain\kotlin\ru\wrenge\keyword\credentials\ClassicCredentialsInfo.kt:

```
package ru.wrenge.keyword.credentials

class ClassicCredentialsInfo(
    id: Long,
    identifier: String,
    login: String,
    override var password: String,
    override var ctime: Long,
    override var mtime: Long
) : CredentialsIdentity(id, identifier, login), ICredentialsInfo {
    constructor(
        identifier: String,
        login: String,
        password: String,
    ) : this(0, identifier, login, password, 0, 0)

    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (!super.equals(other)) return false

        if (other is ICredentialsInfo) {
            if (password != other.password) return false
            if (ctime != other.ctime) return false
            if (mtime != other.mtime) return false
        }

        return true
    }

    override fun hashCode(): Int {
        var result = super.hashCode()
        result = 31 * result + password.hashCode()
        result = 31 * result + ctime.hashCode()
        result = 31 * result + mtime.hashCode()
        return result
    }
}
```

modules\credentials\src\commonMain\kotlin\ru\wrenge\keyword\credentials\CredentialsIdentity.kt:

```
package ru.wrenge.keyword.credentials

open class CredentialsIdentity(
    override var id: Long,
    override var identifier: String,
    override var login: String,
) : ICredentialsIdentity {
    {
        override fun equals(other: Any?): Boolean {
            if (this === other) return true
```

```

        if (other !is ICredentialsIdentity) return false

        if (id != other.id) return false
        if (identifier != other.identifier) return false
        if (login != other.login) return false

        return true
    }

    override fun hashCode(): Int {
        var result = id.hashCode()
        result = 31 * result + identifier.hashCode()
        result = 31 * result + login.hashCode()
        return result
    }
}

```

modules\credentials\src\commonMain\kotlin\ru\wrenge\keyword\credentials\ICredentialsIdentity.kt:

```

package ru.wrenge.keyword.credentials

interface ICredentialsIdentity {
    var id : Long
    var identifier : String
    var login : String
}

```

modules\credentials\src\commonMain\kotlin\ru\wrenge\keyword\credentials\ICredentialsInfo.kt:

```

package ru.wrenge.keyword.credentials

interface ICredentialsInfo : ICredentialsIdentity {
    var password : String
    var ctime : Long
    var mtime : Long
}

```

modules\credentials\src\commonTest\kotlin\ru.wrenge.keyword.credentials\TestCredentials.kt:

```

package ru.wrenge.keyword.credentials

import kotlin.test.Test
import kotlin.test.assertEquals
import kotlin.test.assertNotEquals

class TestCredentials {
    @Test
    fun testIdentities() {
        val id1 = CredentialsIdentity(1, "www.kotlin.ru", "id1")
        val id2 = CredentialsIdentity(2, "www.kotlin.ru", "id2")
        val id3 = CredentialsIdentity(1, "www.kotlin.ru", "id1")
    }
}

```

```

        assertEquals(id1, id1)
        assertEquals(id1.hashCode(), id1.hashCode())
        assertNotEquals(id1, id2)
        assertNotEquals(id1.hashCode(), id2.hashCode())
        assertEquals(id1, id3)
        assertEquals(id1.hashCode(), id3.hashCode())
    }

    @Test
    fun testInfos() {
        val id1 = ClassicCredentialsInfo( "www.kotlin.ru", "id1", "1234")
        val id2 = ClassicCredentialsInfo("www.kotlin.ru", "id1", "1235")
        val id3 = ClassicCredentialsInfo("www.kotlin.ru", "id1", "1234")

        assertEquals(id1, id1)
        assertEquals(id1.hashCode(), id1.hashCode())
        assertNotEquals(id1, id2)
        assertNotEquals(id1.hashCode(), id2.hashCode())
        assertEquals(id1, id3)
        assertEquals(id1.hashCode(), id3.hashCode())
    }

    @Test
    fun testBoth() {
        val id1 = CredentialsIdentity(0, "www.kotlin.ru", "id1")
        val id2 = CredentialsIdentity(0, "www.kotlin.ru", "id2")
        val id3 = ClassicCredentialsInfo("www.kotlin.ru", "id1", "1234")

        assertEquals(id1, id1)
        assertEquals(id1.hashCode(), id1.hashCode())
        assertNotEquals(id1, id2)
        assertNotEquals(id1.hashCode(), id2.hashCode())
        assertEquals(id1, id3)
        assertEquals(id3, id1)
        assertNotEquals(id2.hashCode(), id3.hashCode())
        assertNotEquals(id3.hashCode(), id2.hashCode())
    }
}

```

```

modules\crypt\src\commonMain\kotlin\ru\wrenge\keyword\crypt\AESCryptBase.kt:
package ru.wrenge.keyword.crypt

```

```

import com.soywiz.krypto.AES
import com.soywiz.krypto.Padding
import com.soywiz.krypto.encoding.hex
import com.soywiz.krypto.encoding.unhex

```

```

abstract class AESCryptBase : ICrypt {
    protected abstract val padding : Padding

    override fun encrypt(msg: String, key: String): String {
        val msgBytes = msg.encodeToByteArray().hex.encodeToByteArray()
    }
}

```

```

        val keyBytes = SHA256().digest(key).encodeToByteArray()
        val cypherBytes = encrypt(msgBytes, keyBytes)
        return cypherBytes.hex
    }

    override fun decrypt(msg: String, key: String): String {
        val msgBytes = msg.unhex
        val keyBytes = SHA256().digest(key).encodeToByteArray()

        try {
            val decrypted = decrypt(msgBytes, keyBytes)
            return decrypted.decodeToString().unhex.decodeToString()
        }
        catch (e: Throwable){
            throw IllegalArgumentException(msg, key)
        }
    }

    private fun encrypt(msg: ByteArray, key: ByteArray) : ByteArray {
        return AES.encryptAesEcb(msg, key, padding)
    }

    private fun decrypt(msg: ByteArray, key: ByteArray) : ByteArray {
        return AES.decryptAesEcb(msg, key, padding)
    }
}

```

`modules\crypt\src\commonMain\kotlin\ru\wrenge\keeword\crypt\AESCryptISO.kt:`
`package ru.wrenge.keeword.crypt`

```

import com.soywiz.krypto.Padding

class AESCryptISO : AESCryptBase() {
    override val padding = Padding.ISO10126Padding
}

```

`modules\crypt\src\commonMain\kotlin\ru\wrenge\keeword\crypt\AESCryptZero.kt:`
`package ru.wrenge.keeword.crypt`

```

import com.soywiz.krypto.Padding

class AESCryptZero : AESCryptBase() {
    override val padding = Padding.ZeroPadding
}

```

`modules\crypt\src\commonMain\kotlin\ru\wrenge\keeword\crypt\CRC32.kt:`
`package ru.wrenge.keeword.crypt`

```

class CRC32 : IHasher {
    override fun digest(msg: String): String {
        TODO("Not yet implemented")
    }
}

```



```

@Test
fun testAESRightKey() {
    val cypher = AESCryptZero().encrypt(msg, key)
    assertEquals(msg, AESCryptZero().decrypt(cypher, key))
}

@Test
fun testAESWrongKey() {
    val cypher = AESCryptZero().encrypt(msg, key)
    assertFailsWith<IllegalKeyException> { AESCryptZero().decrypt(cypher, k
ey2) }
}
}

```

```

modules\storage-
base\src\commonMain\kotlin\ru\wreng\keyword\storage\base\IStorageController.kt:
package ru.wreng.keyword.storage.base

import ru.wreng.keyword.credentials.ICredentialsIdentity
import ru.wreng.keyword.credentials.ICredentialsInfo

interface IStorageController {
    fun setEncryptionKey(newKey : String)
    fun resetEncryptionKey()
    fun getIdentities() : StorageReadResult<List<ICredentialsIdentity>>
    fun getInfo(identity: ICredentialsIdentity) : StorageReadResult<ICredentialsInfo>
    fun saveInfo(info : ICredentialsInfo) : StorageAccessResult
    fun updateInfo(info : ICredentialsInfo) : StorageAccessResult
    fun deleteInfo(info : ICredentialsInfo) : StorageAccessResult
    fun clearInfos() : StorageAccessResult
    fun isAccessGranted() : Boolean
    fun close()
}

```

```

modules\storage-
base\src\commonMain\kotlin\ru\wreng\keyword\storage\base\StorageAccessDenyReason
.kt:
package ru.wreng.keyword.storage.base

enum class StorageAccessDenyReason {
    NONE,
    ACCESS_DENIED,
    NOT_FOUND,
    CONFLICT,
    OTHER
}

```

```

modules\storage-
base\src\commonMain\kotlin\ru\wreng\keyword\storage\base\StorageAccessResult.kt:
package ru.wreng.keyword.storage.base

```

```

open class StorageAccessResult(
    val denyReason : StorageAccessDenyReason = StorageAccessDenyReason.NONE,
    val denyMsg : String = "") {
    open val isSuccess : Boolean
        get() = denyReason == StorageAccessDenyReason.NONE
}

```

modules\storage-

base\src\commonMain\kotlin\ru\wrenge\keyword\storage\base\StorageReadResult.kt:

```

package ru.wrenge.keyword.storage.base

```

```

class StorageReadResult<T>(
    val value: T?,
    denyReason: StorageAccessDenyReason = StorageAccessDenyReason.NONE,
    denyMsg: String = ""
) : StorageAccessResult(denyReason, denyMsg) {
    override val isSuccess: Boolean
        get() = super.isSuccess && value != null

    constructor(denyReason: StorageAccessDenyReason, denyMsg: String) : this(null, denyReason, denyMsg)
}

```

modules\storage-

mobile\src\androidMain\kotlin\ru.wrenge.keyword.storage.mobile\StorageController.kt:

```

package ru.wrenge.keyword.storage.mobile

import android.content.Context
import com.squareup.sqldelight.android.AndroidSqliteDriver
import com.squareup.sqldelight.db.SqlDriver

internal actual class StorageController(private val context : Context) : BaseStorageController() {
    override fun createDriver(): SqlDriver {
        return AndroidSqliteDriver(Storage.Schema, context, dbname)
    }
}

```

modules\storage-

mobile\src\androidTest\kotlin\ru.wrenge.keyword.storage.mobile\DBControllerTest.kt:

```

package ru.wrenge.keyword.storage.mobile

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import androidx.test.core.app.ApplicationProvider
import org.junit.runner.RunWith
import org.robolectric.RobolectricTestRunner
import java.io.File

@RunWith(RobolectricTestRunner::class)

```

```

actual class DBControllerTest : DBControllerTestBase() {
    private val context: Context by lazy {
        ApplicationProvider.getApplicationContext()
    }

    actual override fun deleteDatabase() : Boolean {
        return SQLiteDatabase.deleteDatabase(File("${context.dataDir}/databases
/${context.dbname}"))
    }

    actual override fun initStorage() {
        controller = StorageController(context)
    }
}

```

modules\storage-

mobile\src\commonMain\kotlin\ru.wreng.keeword.storage.mobile\BaseStorageController.kt:

```

package ru.wreng.keeword.storage.mobile

import com.squareup.sqldelight.db.SqlDriver
import ru.wreng.keeword.credentials.*
import ru.wreng.keeword.crypt.AESCryptISO
import ru.wreng.keeword.storage.base.*

abstract class BaseStorageController : IStorageController {
    internal val dbname = "keeword-storage.db"
    private var encryptionKey: String? = null
    private val crypt = AESCryptISO()

    private var _driver : SqlDriver? = null
    private val driver: SqlDriver
        get() {
            if(_driver == null)
                _driver = createDriver()
            return _driver!!
        }

    private var _storage: Storage? = null
    private val storage: Storage
        get() {
            if(_storage == null)
                _storage = Storage(driver)
            return _storage!!
        }

    protected abstract fun createDriver(): SqlDriver

    override fun setEncryptionKey(newKey: String) {
        encryptionKey = newKey
    }
}

```

```

override fun resetEncryptionKey() {
    encryptionKey = null
}

override fun getIdentities(): StorageReadResult<List<ICredentialsIdentity>>
{
    if (encryptionKey == null) {
        return StorageReadResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }
    val list = storage.credentialsQueries.selectAllIdentities { id, identifier, login ->
        decryptCredentialsIdentity(CredentialsIdentity(id, identifier, login))
    }.executeAsList()

    return StorageReadResult(list)
}

override fun getInfo(identity: ICredentialsIdentity): StorageReadResult<ICredentialsInfo> {
    if (encryptionKey == null) {
        return StorageReadResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

    if (identity.id == 0L) {
        return StorageReadResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity not saved"
        )
    }

    val value =
        doGetInfo(identity)
        ?: return StorageReadResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity does not exist"
        )

    return StorageReadResult(decryptCredentialsInfo(value))
}

override fun saveInfo(info: ICredentialsInfo): StorageAccessResult {
    if (encryptionKey == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

```

```

    )
}

if (doGetInfo(info) != null) {
    return StorageAccessResult(
        StorageAccessDenyReason.CONFLICT,
        "Credentials info with id ${info.id} already exists"
    )
}

val eInfo = encryptCredentialsInfo(info)
val savedInstance : ICredentialsInfo? = storage.credentialsQueries.transactionWithResult {
    storage.credentialsQueries.insertNewCredentials(eInfo.identifier, eInfo.login, eInfo.password)
    val newID = storage.credentialsQueries.selectLastInsertedId().executeAsOne()
    doGetInfo(CredentialsIdentity(newID, eInfo.identifier, eInfo.login)
)
}

if (savedInstance != null) {
    info.id = savedInstance.id
    info.ctime = savedInstance.ctime
    info.mtime = savedInstance.mtime
}

return StorageAccessResult()
}

override fun updateInfo(info: ICredentialsInfo): StorageAccessResult {
    if (encryptionKey == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.ACCESS_DENIED,
            "Key is not set"
        )
    }

    if (info.id == 0L) {
        return StorageAccessResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Identity not saved"
        )
    }

    if (doGetInfo(info) == null) {
        return StorageAccessResult(
            StorageAccessDenyReason.NOT_FOUND,
            "Credentials info with id ${info.id} already exists"
        )
    }
}

```

```

        val eInfo = encryptCredentialsInfo(info)
        storage.credentialsQueries.transaction {
            storage.credentialsQueries.updateCredentials(
                eInfo.identifier,
                eInfo.login,
                eInfo.password,
                eInfo.ctime,
                eInfo.id
            )
            info.mtime = doGetInfo(eInfo)?.mtime ?: 0
        }

        return StorageAccessResult()
    }

    override fun deleteInfo(info: ICredentialsInfo): StorageAccessResult {
        if (encryptionKey == null) {
            return StorageAccessResult(
                StorageAccessDenyReason.ACCESS_DENIED,
                "Key is not set"
            )
        }

        if (info.id == 0L) {
            return StorageAccessResult(
                StorageAccessDenyReason.NOT_FOUND,
                "Identity not saved"
            )
        }

        if (doGetInfo(info) == null) {
            return StorageAccessResult(
                StorageAccessDenyReason.NOT_FOUND,
                "Credentials info with id ${info.id} does not exist"
            )
        }

        storage.credentialsQueries.deleteCredentials(info.id)

        return StorageAccessResult()
    }

    override fun clearInfos(): StorageAccessResult {
        if (encryptionKey == null) {
            return StorageAccessResult(
                StorageAccessDenyReason.ACCESS_DENIED,
                "Key is not set"
            )
        }

        storage.credentialsQueries.clearCredentials()
    }

```

```

        return StorageAccessResult()
    }

    override fun isAccessGranted(): Boolean {
        TODO("Not yet implemented")
    }

    private fun doGetInfo(identity: ICredentialsIdentity): ClassicCredentialsInfo? {
        return if(identity.id > 0)
            storage.credentialsQueries.selectCredentials(identity.id, ::ClassicCredentialsInfo).executeAsOneOrNull()
        else {
            val encryptedIdentity = encryptCredentialsIdentity(identity)
            storage.credentialsQueries.selectCredentialsByIdentity(
                encryptedIdentity.identifier,
                encryptedIdentity.login,
                ::ClassicCredentialsInfo
            ).executeAsOneOrNull()
        }
    }

    private fun decryptCredentialsIdentity(identity: ICredentialsIdentity): ICredentialsIdentity {
        val key = encryptionKey!!
        val dIdentifier = crypt.decrypt(identity.identifier, key)
        val dLogin = crypt.decrypt(identity.login, key)
        return CredentialsIdentity(identity.id, dIdentifier, dLogin)
    }

    private fun encryptCredentialsIdentity(identity: ICredentialsIdentity): ICredentialsIdentity {
        val key = encryptionKey!!
        val eIdentifier = crypt.encrypt(identity.identifier, key)
        val eLogin = crypt.encrypt(identity.login, key)
        return CredentialsIdentity(identity.id, eIdentifier, eLogin)
    }

    private fun decryptCredentialsInfo(info: ICredentialsInfo): ICredentialsInfo {
        val key = encryptionKey!!
        val identity = decryptCredentialsIdentity(info)
        val dPassword = crypt.decrypt(info.password, key)
        return ClassicCredentialsInfo(info.id, identity.identifier, identity.login, dPassword, info.ctime, info.mtime)
    }

    private fun encryptCredentialsInfo(
        info: ICredentialsInfo
    ): ICredentialsInfo {
        val key = encryptionKey!!
        val identity = encryptCredentialsIdentity(info)

```

```

        val ePassword = crypt.encrypt(info.password, key)
        return ClassicCredentialsInfo(info.id, identity.identifier, identity.lo
gin, ePassword, info.ctime, info.mtime)
    }

    override fun close() {
        driver.close()
        _driver = null
        _storage = null
    }
}

```

modules\storage-
mobile\src\commonMain\kotlin\ru.wreng.keeword.storage.mobile\StorageController.k
t:

```

package ru.wreng.keeword.storage.mobile

internal expect class StorageController : BaseStorageController

```

modules\storage-
mobile\src\commonTest\kotlin\ru.wreng.keeword.storage.mobile\DBControllerTest.kt
:

```

package ru.wreng.keeword.storage.mobile

expect class DBControllerTest : DBControllerTestBase {
    override fun deleteDatabase(): Boolean
    override fun initStorage()
}

```

modules\storage-
mobile\src\commonTest\kotlin\ru.wreng.keeword.storage.mobile\DBControllerTestBas
e.kt:

```

package ru.wreng.keeword.storage.mobile

import ru.wreng.keeword.credentials.ClassicCredentialsInfo
import ru.wreng.keeword.credentials.ICredentialsInfo
import ru.wreng.keeword.storage.base.StorageAccessDenyReason
import ru.wreng.keeword.storage.base.StorageAccessResult
import ru.wreng.keeword.storage.base.StorageReadResult
import kotlin.test.*

abstract class DBControllerTestBase {
    lateinit var controller: BaseStorageController
    protected val key: String = "test123"

    @BeforeTest
    fun init() {
        initStorage()
        setEncryptionKey()
    }

    private fun setEncryptionKey() {

```

```

        controller.setEncryptionKey(key)
    }

    private fun resetEncryptionKey() {
        controller.resetEncryptionKey()
    }

    @AfterTest
    fun defer() {
        controller.close()
        deleteDatabase()
    }

    abstract fun deleteDatabase(): Boolean
    abstract fun initStorage()

    @Test
    fun testAccessNoKey() {
        resetEncryptionKey()
        val credInfo = ClassicCredentialsInfo(
            "www.kotlin.com", "testAccessNoKey", "1234"
        )

        var resp: StorageAccessResult
        resp = controller.getIdentities()
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)

        resp = controller.getInfo(credInfo)
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)

        resp = controller.saveInfo(credInfo)
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)

        resp = controller.updateInfo(credInfo)
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)

        resp = controller.deleteInfo(credInfo)
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)

        resp = controller.clearInfos()
        assertFalse(resp.isSuccess)
        assertEquals(StorageAccessDenyReason.ACCESS_DENIED, resp.denyReason)
    }

    @Test
    fun testGetInfo() {
        val credInfo1 = ClassicCredentialsInfo(

```

```

        "www.kotlin.com", "testGetInfo1", "1234"
    )

    val credInfo2 = ClassicCredentialsInfo(
        "www.kotlin.com", "testGetInfo2", "1234"
    )

    controller.saveInfo(credInfo1)
    var result = controller.getInfo(credInfo2)
    assertFalse(result.isSuccess)
    assertEquals(StorageAccessDenyReason.NOT_FOUND, result.denyReason)

    credInfo2.id = 2
    result = controller.getInfo(credInfo2)
    assertFalse(result.isSuccess)
    assertEquals(StorageAccessDenyReason.NOT_FOUND, result.denyReason)
}

@Test
fun testSaveInfo() {
    val credInfo = ClassicCredentialsInfo(
        "www.kotlin.com", "testSaveInfo", "1234"
    )

    var resp: StorageAccessResult = controller.saveInfo(credInfo)
    assertTrue(resp.isSuccess, resp.denyMsg)
    assertEquals(0, credInfo.id, "testSaveInfo id = 0")

    val result: StorageReadResult<ICredentialsInfo> = controller.getInfo(credInfo)
    assertTrue(result.isSuccess, result.denyMsg)
    assertEquals(credInfo, result.value, "found testSaveInfo is not the same")

    resp = controller.saveInfo(credInfo)
    assertFalse(resp.isSuccess, "saved duplicate testSaveInfo")
    assertEquals(StorageAccessDenyReason.CONFLICT, resp.denyReason, resp.denyMsg)
}

@Test
fun testUpdateInfo() {
    val credInfo1 = ClassicCredentialsInfo(
        "www.kotlin.com", "testUpdateInfo1", "1234"
    )

    val credInfo2 = ClassicCredentialsInfo(
        "www.kotlin.com", "testUpdateInfo2", "1234"
    )

    assertEquals(credInfo1, credInfo2)

    controller.saveInfo(credInfo1)

```

```

var resp = controller.updateInfo(credInfo2)
assertFalse(resp.isSuccess)
assertEquals(StorageAccessDenyReason.NOT_FOUND, resp.denyReason)

credInfo2.id = 2
resp = controller.updateInfo(credInfo2)
assertFalse(resp.isSuccess)
assertEquals(StorageAccessDenyReason.NOT_FOUND, resp.denyReason)

credInfo2.id = credInfo1.id
resp = controller.updateInfo(credInfo2)
assertTrue(resp.isSuccess, resp.denyMsg)

val result = controller.getInfo(credInfo2)
assertTrue(result.isSuccess, result.denyMsg)
assertEquals(credInfo2, result.value)
assertNotEquals(credInfo1, result.value)
}

@Test
fun testDeleteInfo() {
    val credInfo1 = ClassicCredentialsInfo(
        "www.kotlin.com", "testDeleteInfo1", "1234"
    )

    val credInfo2 = ClassicCredentialsInfo(
        "www.kotlin.com", "testDeleteInfo2", "1234"
    )

    assertNotEquals(credInfo1, credInfo2)

    controller.saveInfo(credInfo1)

    var resp = controller.deleteInfo(credInfo2)
    assertFalse(resp.isSuccess)
    assertEquals(StorageAccessDenyReason.NOT_FOUND, resp.denyReason)

    credInfo2.id = 2
    resp = controller.deleteInfo(credInfo2)
    assertFalse(resp.isSuccess)
    assertEquals(StorageAccessDenyReason.NOT_FOUND, resp.denyReason)

    resp = controller.deleteInfo(credInfo1)
    assertTrue(resp.isSuccess, resp.denyMsg)

    val result = controller.getInfo(credInfo1)
    assertFalse(result.isSuccess)
    assertEquals(StorageAccessDenyReason.NOT_FOUND, result.denyReason)
}

@Test

```

```

fun testClearInfos() {
    val credInfo1 = ClassicCredentialsInfo(
        "www.kotlin.com", "testClearInfos1", "1234"
    )

    val credInfo2 = ClassicCredentialsInfo(
        "www.kotlin.com", "testClearInfos2", "1234"
    )

    assertEquals(credInfo1, credInfo2)

    controller.saveInfo(credInfo1)
    controller.saveInfo(credInfo2)

    var result = controller.getIdentities()
    assertTrue(result.isSuccess, result.denyMsg)
    assertEquals(2, result.value?.count() ?: 0)

    val resp = controller.clearInfos()
    assertTrue(resp.isSuccess, result.denyMsg)

    result = controller.getIdentities()
    assertTrue(result.isSuccess, result.denyMsg)
    assertEquals(0, result.value?.count() ?: 0)
}

@Test
fun testDeleteDB() {
    val credInfo1 = ClassicCredentialsInfo(
        "www.kotlin.com", "testDeleteDB1", "1234"
    )

    val credInfo2 = ClassicCredentialsInfo(
        "www.kotlin.com", "testDeleteDB2", "1234"
    )

    assertEquals(credInfo1, credInfo2)

    controller.saveInfo(credInfo1)
    controller.saveInfo(credInfo2)

    var result = controller.getIdentities()
    assertTrue(result.isSuccess, result.denyMsg)
    assertEquals(2, result.value?.count() ?: 0)

    controller.close()
    assertTrue(deleteDatabase())

    result = controller.getIdentities()
    assertTrue(result.isSuccess, result.denyMsg)
    assertEquals(0, result.value?.count() ?: 0)
}

```

```
}
```

```
modules\storage-
mobile\src\iosMain\kotlin\ru.wrenge.keeword.storage.mobile\StorageController.kt:
package ru.wrenge.keeword.storage.mobile
```

```
import com.squareup.sqldelight.db.SqlDriver
import com.squareup.sqldelight.drivers.native.NativeSqliteDriver
```

```
internal actual class StorageController : BaseStorageController() {
    override fun createDriver(): SqlDriver {
        return NativeSqliteDriver(Storage.Schema, dbname)
    }
}
```

```
modules\storage-
mobile\src\iosTest\kotlin\ru.wrenge.keeword.storage.mobile\DBControllerTest.kt:
package ru.wrenge.keeword.storage.mobile
```

```
import co.touchlab.sqliter.DatabaseFileContext
```

```
actual class DBControllerTest : DBControllerTestBase() {
    actual override fun deleteDatabase(): Boolean {
        DatabaseFileContext.deleteDatabase(controller.dbname)
        return true
    }

    actual override fun initStorage() {
        controller = StorageController()
    }
}
```

```
modules\storage-
mobile\src\commonMain\sqldelight\ru.wrenge.keeword.storage.mobile\credentials.sql:
```

```
CREATE TABLE credentials
(
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    identifier TEXT NOT NULL DEFAULT "",
    login TEXT NOT NULL DEFAULT "",
    password TEXT NOT NULL DEFAULT "",
    ctime INTEGER NOT NULL,
    mtime INTEGER NOT NULL DEFAULT 0
);
```

```
CREATE INDEX credentials_id ON credentials(id);
CREATE INDEX credentials_domain ON credentials(identifier);
```

```
selectAll:
SELECT id, identifier, login, password
FROM credentials;
```

```
selectAllIdentities:
```

```
SELECT id, identifier, login
FROM credentials;
```

```
selectLastInsertedId:
SELECT last_insert_rowid();
```

```
selectCredentials:
SELECT *
FROM credentials
WHERE id = ?
LIMIT(1);
```

```
selectCredentialsByIdentity:
SELECT *
FROM credentials
WHERE identifier = ? AND login = ?
LIMIT(1);
```

```
insertNewCredentials:
INSERT INTO credentials(identifier, login, password, ctime)
VALUES (?, ?, ?, (SELECT DATE('now')));
```

```
insertExistingCredentials:
INSERT INTO credentials(id, identifier, login, password, ctime, mtime)
VALUES (?, ?, ?, ?, ?, ?);
```

```
deleteCredentials:
DELETE
FROM credentials
WHERE id = ?;
```

```
updateCredentials:
UPDATE credentials
SET identifier = ?,
    login = ?,
    password = ?,
    ctime = ?,
    mtime = (SELECT DATE('now'))
WHERE id = ?;
```

```
clearCredentials:
DELETE FROM credentials;
```