МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

МЕНЕДЖЕР ПАРОЛЕЙ

Студент _____ Лобанов Д.В.
(подпись, дата)

Руководитель _____ Митрохин М.А.
(подпись, дата)

Нормоконтролёр _____ А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 02.06.2023 г. №14)

Заведующий кафедрой _____ М.А. Митрохин
(подпись)

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

Секретарь ГЭК _____ Е.И. Гурин
(подпись)

Пенза 2023

Реферат

Бакалаврская работа содержит 100 л., 25 рис., 1 табл., 5 источн., 3 прил.

МЕНЕДЖЕР ПАРОЛЕЙ, ШИФРОВАНИЕ, *JSON*, *WINDOWS FORMS*

Цель работы – разработать программу, позволяющую безопасно хранить пароли.

Объектом разработки является приложение, позволяющее экспортировать, импортировать и изменять базу паролей. Приложение должно обладать графическим пользовательским интерфейсом, иметь возможность объединять пароли в группы, сохранять пароли в зашифрованный файл, экспортировать зашифрованные файлы и дешифровать их, а также генерировать новые пароли.

Разработка проводилась в среде программирования *Visual Studio* 2022. В качестве языка программирования был выбран *C#*.

В результате проведенной работы разработано приложение, удовлетворяющее поставленной задаче, а именно: удобный пользовательский интерфейс, возможность редактирования паролей, импорт и экспорт базы паролей в файл, шифрование и дешифрация файла, генерация новых паролей.

Содержание

Перечень сокращений и обозначений

.NET – кроссплатформенная платформа для создания различных типов приложений

JSON – Текстовый формат обмена данными, основанный на *JavaScript*

MVC (*Model-View-Controller*) – Схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер

UML (*Unified Modeling Language*) – унифицированный язык моделирования

ГОСТ – региональный стандарт, принятый Межгосударственным советом по стандартизации, метрологии и сертификации Содружества Независимых Государств

ПЭВМ – персональная электронно-вычислительная машина

ПК – персональный компьютер

Форма – графический элемент пользовательского интерфейса, который представляет собой окно или контейнер, предназначенный для отображения и взаимодействия с пользователем

Фреймворк – программная платформа, которая упрощает разработку программного продукта, определяет структуру проекта

Введение

В настоящее время разработка программного обеспечения является важным элементом современного мира. С каждым днем число пользователей, использующих интернет-сервисы, увеличивается, и вместе с ним растут их требования к качеству и безопасности обработки и хранения персональных данных.

Одним из наиболее важных аспектов безопасности является сохранение конфиденциальности паролей пользователей.

Для хорошего пароля, устойчивого к атакам рекомендуется следовать большому количеству правил:

- длина. Пароль должен быть достаточно длинным. Рекомендуется использовать пароли длиной не менее 12 символов;

- разнообразие символов. Пароль должен состоять из разнообразных символов, включая прописные и строчные буквы, цифры и специальные символы например(^, !, @, #, $);

- неиспользование личной информации. Необходимо избегать использования личной информации, такой как имена, даты рождения, адреса и т.д., которую можно легко угадать или получить из публичных источников;

- неиспользование последовательностей. Необходимо избегать использования последовательностей символов или цифр, таких как "123456" или "qwerty", которые могут быть легко угаданы;

- уникальность. Рекомендуется использовать уникальные пароли для разных учетных записей. Использование одного и того же пароля для разных сервисов может повлечь серьезные последствия в случае утечки пароля.

В данной работе представлено приложение, разработанное для безопасного хранения паролей рядового пользователя. Оно позволяет сохранять пароли в зашифрованном виде и предоставляет возможность удобного редактирования, добавления и удаления сохраненных паролей, а также для генерирования новых паролей.

1 Менеджер паролей

Актуальность данной темы обусловлена тем, что в настоящее время важно обеспечить безопасность личной информации и конфиденциальных данных. Многие из пользователей сталкиваются с проблемой запоминания множества сложных паролей для каждого сервиса, и поэтому для ставят один и тот же пароль, что в разы облегчает задачу злоумышленнику, если он захочет получить контроль над каким-либо аккаунтом.

Если пользователь хочет обеспечить безопасность своих аккаунтов в сети, ему необходимо использовать разные пароли к разным аккаунтам. Ставится вопрос о хранении данных аккаунта где-либо и о фантазии придумывания стойких паролей. Есть несколько способов хранения.

Можно попросту писать пароль в блокнотик. В таком случае, около компьютера пользователя будет лежать блокнот, в котором хаотично записаны какие-либо аккаунты, отдельные пароли, номера телефонов вперемешку с записями. Также однажды он может потеряться, и в таком случае необходимо будет заново восстанавливать доступ ко всем записанным туда аккаунтам.

Можно хранить в текстовом файле на компьютере. Этот файлик всегда под рукой, его не нужно искать, и можно просто скопировать с какой-либо страницы твой новый логин-пароль, вставить его в «текстовик» и сохранить. К сожалению, это более опасный вариант, так как если к твоему компьютеру получат доступ, то можно попросту скопировать пароли, или сделать скриншот.

Можно обратиться к существующим сейчас менеджерам паролей. К сожалению, для рядового пользователя компьютера чаще всего даже неизвестно, что такие существуют. Имеющиеся на рынке менеджеры паролей зачастую являются либо платными, либо требуют особые знания и навыки в использовании компьютера, что делает их сложными для рядового пользователя. Поэтому существует потребность в разработке простого и удобного в использовании приложения для хранения паролей.

Благодаря применению современных алгоритмов шифрования, сохраненные пароли остаются надежно защищенными от несанкционированного доступа. Кроме того, приложение обладает простым и интуитивно понятным пользовательским интерфейсом, что делает его использование максимально удобным.

Менеджеры паролей - это решение, которое значительно улучшает безопасность и удобство использования паролей. Одним из главных преимуществ менеджеров паролей является удобство использования. Они предлагают простой и интуитивно понятный пользовательский интерфейс, который позволяет без труда создавать, хранить и управлять паролями. Больше не нужно запоминать множество паролей или записывать их на бумаге, что может представлять угрозу для безопасности. Менеджеры паролей автоматически заполняют поля входа на веб-сайтах и приложениях, облегчая процесс аутентификации и сокращая время, затрачиваемое на ввод пароля.

Следующим важным аспектом является безопасность. Менеджеры паролей используют мощные алгоритмы шифрования для защиты ваших паролей и конфиденциальных данных. Все пароли хранятся в зашифрованном виде и доступны только вам с использованием основного пароля или другого подтверждающего фактора. Это предотвращает несанкционированный доступ и утечки данных, обеспечивая высокий уровень безопасности.

Менеджеры паролей также способствуют созданию сильных паролей. Они предлагают генераторы паролей, которые создают случайные и сложные комбинации символов, невозможные для угадывания или взлома. Это помогает предотвратить использование слабых паролей, которые могут стать легкой целью для злоумышленников.

На рисунке 1.1 показаны варианты использования классического менеджера паролей.

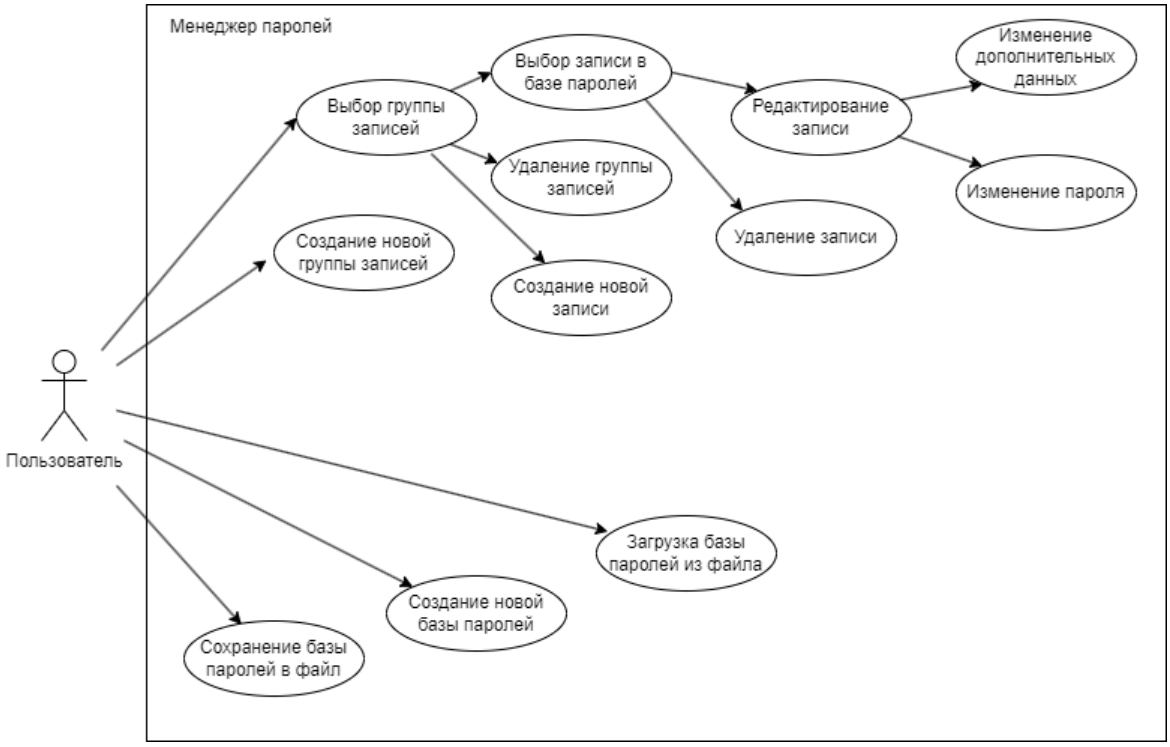


Рисунок 1.1 – *UML*-диаграмма вариантов использования менеджера паролей

1.1 Анализ существующих аналогов

Среди аналогов систем для управления паролями, можно выделить следующие:

- *KeePass* – бесплатная и открытая система управления паролями;

- *Pass* – командная строка для управления паролями с использованием шифрования *GPG*;

- *Dashlane* – коммерческая платформа для управления паролями с функциями автозаполнения и синхронизации.

Эти системы предоставляют возможность безопасного хранения, генерации и автоматического заполнения паролей, обеспечивая удобство использования и защиту учетных данных.

1.1.1 Менеджер паролей «*KeePass*»

KeePass – это свободное и бесплатное приложение для управления паролями, которое позволяет сохранять и защищать пароли пользователей в зашифрованном виде. *KeePass* поддерживает множество функций, включая

автоматическое заполнение паролей, возможность генерации паролей и привязки к определенным *URL*-адресам.

KeePass также предоставляет возможность хранения других конфиденциальных данных, таких как номера кредитных карт, *PIN*-коды и т.д. Все данные хранятся в одном зашифрованном файле, который защищен мастер-паролем.

Среди основных преимуществ *KeePass* можно отметить его свободное распространение и открытый исходный код, что позволяет пользователям проверять его безопасность и вносить свои улучшения. *KeePass* также поддерживает множество дополнительных плагинов, которые расширяют его функциональность и позволяют настроить его под конкретные потребности пользователя.

Среди недостатков *KeePass* можно отметить его сложность в использовании для новичков, особенно в настройке и синхронизации между несколькими устройствами. Кроме того, управление множеством паролей может быть трудным для пользователей без определенных навыков. Пример интерфейса программы можно увидеть на рисунке 1.2. *KeePass* – хороший вариант для опытных пользователей, которые не страшатся громоздких интерфейсов, долгой настройки и перегруженности функций.

1.1.2 Менеджер паролей «*Pass*»

Pass – это приложение для хранения паролей и других конфиденциальных данных в зашифрованном виде. Оно основано на командной строке, и позволяет хранить пароли в виде текстовых файлов, зашифрованных с помощью *GPG*.

Среди преимуществ приложения *Pass* можно отметить его открытый исходный код, что позволяет пользователям проверять безопасность кода и убедиться в его надежности. Кроме того, приложение не имеет облачного хранения, что уменьшает риски взлома и утечки данных.

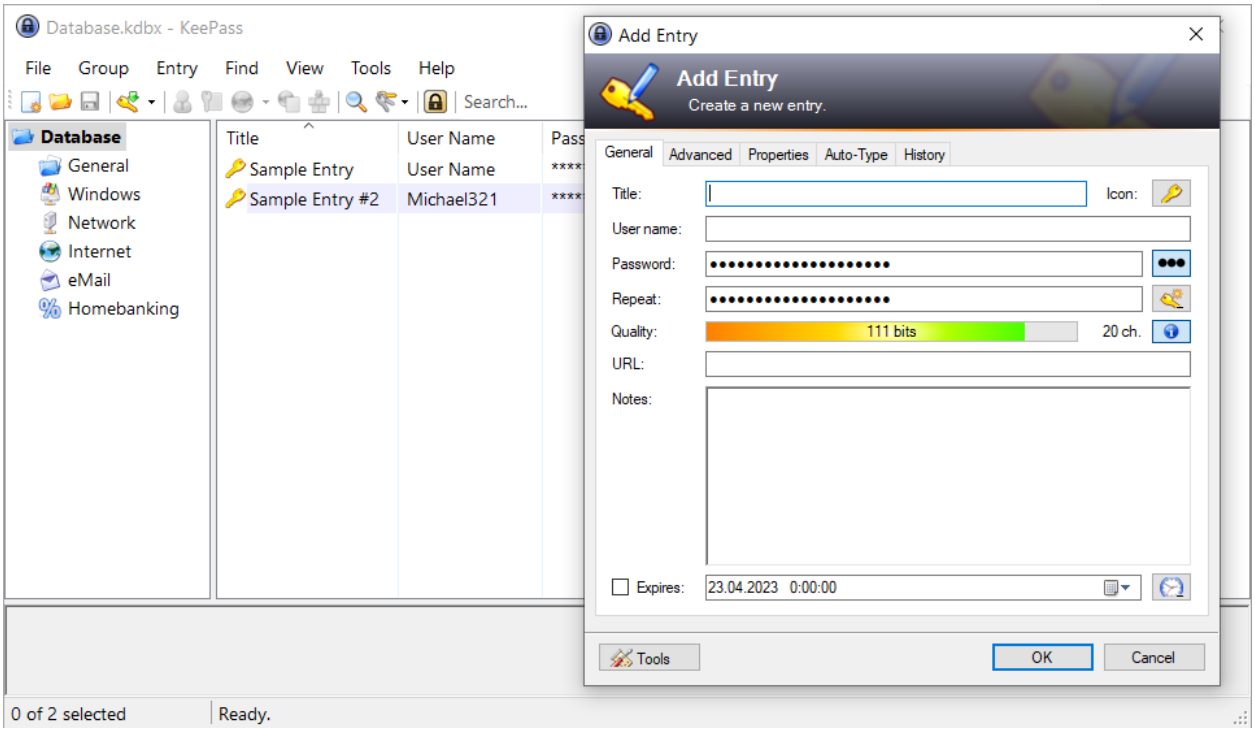


Рисунок 1.2 – Интерфейс программы *KeePass*

Однако, из-за того, что приложение основано на командной строке, оно может показаться сложным для пользователей, не знакомых с этим способом работы. Кроме того, приложение не предоставляет удобный графический интерфейс, что может быть неудобно для пользователей, предпочитающих визуальную навигацию. Пример интерфейса программы показан на рисунке 1.3.

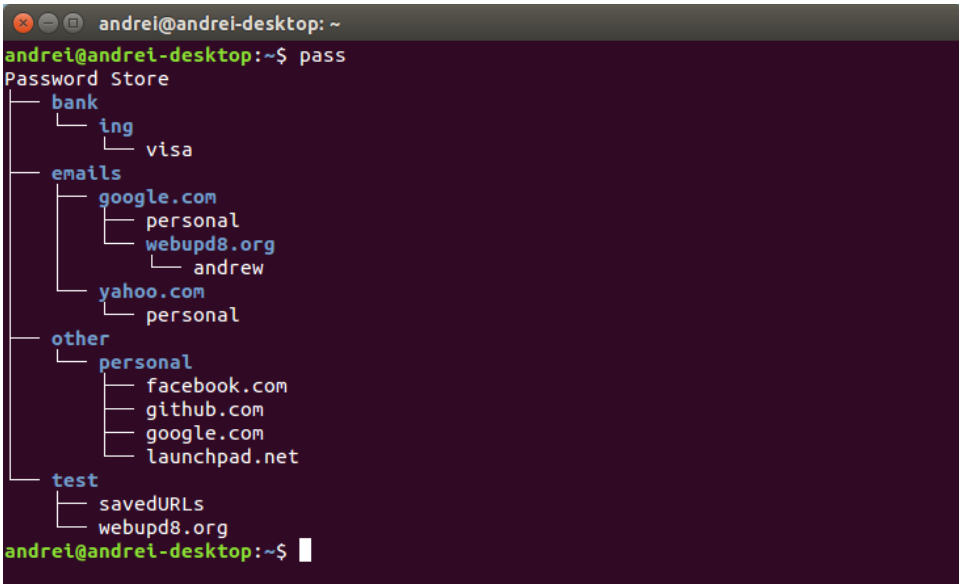


Рисунок 1.3 – Интерфейс программы «*Pass*»

В целом, приложение *Pass* хороший выбор для опытных пользователей, которые предпочитают командную строку и ценят безопасность данных. Однако, для новичков и тех, кто ищет более удобный и простой способ хранения паролей, могут быть лучшие альтернативы с более простым и удобным интерфейсом.

1.1.3 Менеджер паролей «*Dashlane*»

Dashlane - это коммерческая платформа для управления паролями, которая предлагает ряд преимуществ и имеет некоторые недостатки.

Преимущества «*Dashlane*»:

- удобство использования. «*Dashlane*» предоставляет интуитивно понятный интерфейс и интеграцию с веб-браузерами, что облегчает сохранение, заполнение и автоматическое обновление паролей на различных веб-сайтах (рисунок 1.4);

- высокий уровень безопасности. Платформа обеспечивает сильное шифрование паролей и других конфиденциальных данных, а также предлагает функции, такие как двухфакторная аутентификация и хранение данных в зашифрованном виде на устройстве.

Недостатки «*Dashlane*»:

– платная модель подписки. «*Dashlane*» предлагает платную модель подписки для полного доступа ко всем функциям и синхронизации между устройствами. Бесплатная версия ограничена в функциональности;

– зависимость от облачного хранения. Для синхронизации данных и доступа к ним с разных устройств «*Dashlane*» использует облачное хранилище, что может вызывать определенные опасения с точки зрения конфиденциальности данных.

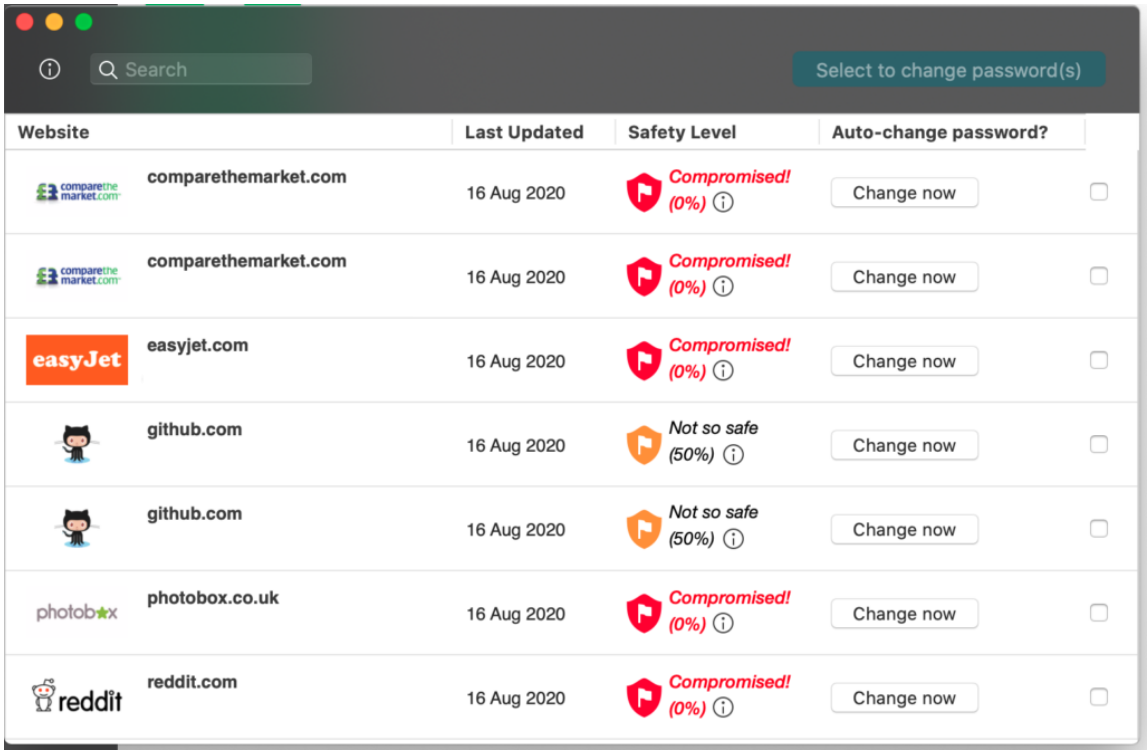


Рисунок 1.4 – Интерфейс программы «*Dashlane*»

1.2 Постановка задачи

Требуется разработать приложение, которое должно соответствовать следующему функционалу:

– многооконное приложение;

– графический интерфейс;

– безопасность оперативной памяти приложения;

– возможность добавлять записи в базу паролей;

– возможность добавления дополнительной информации в запись кроме пароля (логин, название аккаунта, привязанный номер телефона и пр.);

– возможность редактирования записей в базе паролей;

– возможность поиска записей в базе паролей;

– возможность объединять записи в группы;

– возможность удаления записей из базы паролей;

– возможность удаления групп записей из базы паролей;

– импорт в файл и экспорт из файла базы паролей;

– автономное хранение базы паролей в файле на компьютере или съёмном носителе с использованием современных алгоритмов шифрования;

– встроенный инструмент генерирования криптографически стойких паролей.

Технические требования для запуска приложения:

– *Windows* 7 или выше;

– *Microsoft .NET Framework* 4.7.1 или выше;

– Свободное место на *HDD*: 50 Мбайт;

– Встроенное графическое ядро.

1.3 Выбор языка программирования

1.3.1 Язык программирования *C++*

C++ – объектно-ориентированный, компилируемый язык программирования. *C++* гораздо ближе к низкоуровневому ассемблеру, чем другие объектно-ориентированные языки, Но *C++* не является достаточно гибким для создания графического интерфейса.

1.3.2 Язык программирования *Java*

Java является главным конкурентом языка *C#*. Этот язык прост в использовании, является кроссплатформенным, а также имеет гибкие настройки для создания пользовательских интерфейсов. К сожалению, бесплатные среды разработки на этом языке менее удобны, чем среда *Visual studio*.

1.3.3 Язык программирования *Python*

Python - это интерпретируемый язык программирования с динамической типизацией, который широко используется в различных областях, включая науку о данных, машинное обучение, веб-разработку, научные вычисления и другие.

Он предоставляет широкий набор стандартных библиотек и фреймворков, что делает его очень гибким и удобным для разработки быстрых прототипов и решений. Также *Python* имеет чистый и простой синтаксис, что делает его доступным для новичков.

Несмотря на множество преимуществ, у *Python* также есть недостатки. В частности, он является интерпретируемым языком, что означает, что он может работать медленнее, чем компилируемые языки. Кроме того, он может иметь проблемы с масштабируемостью, особенно при работе с большими объемами данных.

1.3.4 Язык программирования *C#*

C# разрабатывался *Microsoft* и является одним из основных языков для разработки приложений под платформу *Windows*. Он полностью интегрирован с *.NET Framework*, что позволяет создавать мощные и эффективные приложения.

Этот язык является типизированным языком, что позволяет выявлять ошибки на стадии компиляции. Это позволяет создавать более надежные и безопасные программы.

Одной из главных причин, почему *C#* является отличным выбором для разработки приложений, является удобство создания графических интерфейсов. С помощью *Windows Forms* и *WPF* (*Windows Presentation Foundation*) появляется возможность создавать сложные и красивые пользовательские интерфейсы с минимальным количеством усилий.

1.4 Выводы

По результатам анализа существующих аналогов было принято решение разработать менеджер паролей, который будет решать задачу хранения паролей, а также предоставлять возможность взаимодействия с созданной приложением базой паролей.

Разработка будет производиться с помощью языка *C#*, так как C# позволяет создавать удобный и понятный для пользователя графический интерфейс.

1.5 Выбор инструментов разработки

1.5.1 Выбор среды разработки

В качестве среды разработки было решено выбрать *Microsoft Visual Studio*, которая обладает рядом преимуществ для разработки приложений на платформе

Windows. Одним из ключевых преимуществ является встроенная поддержка системы контроля версий, включая интеграцию с *Git*, что облегчает управление версиями кода. Кроме того, *Visual Studio* предоставляет удобную среду для верстки интерфейса *Windows Forms*, позволяя интуитивно создавать и настраивать пользовательские интерфейсы, размещать элементы управления на форме и задавать им свойства. Это существенно ускоряет процесс разработки и помогает создать эффективный и привлекательный пользовательский интерфейс. Все эти факторы делают *Microsoft Visual Studio* предпочтительным выбором для разработки приложений на платформе Windows.

1.5.2 Выбор платформы для реализации графического интерфейса

В качестве графической платформы приложения была выбрана платформа *Windows Forms*, которая предоставляет мощный и гибкий инструментарий для создания пользовательского интерфейса. *Windows Forms* обладает рядом преимуществ, делающих его удобным выбором для разработки приложений.

Windows Forms обеспечивает простоту и интуитивность разработки интерфейса благодаря использованию визуального дизайнера форм, который позволяет размещать и настраивать элементы управления на форме в удобном графическом режиме. Это упрощает процесс создания пользовательских интерфейсов и сокращает время разработки.

Также *Windows Forms* предлагает широкий набор предопределенных элементов управления, таких как кнопки, текстовые поля, списки, таблицы и другие, что обеспечивает разработчикам гибкость и возможность создания разнообразных пользовательских интерфейсов. Кроме того, *Windows Forms* обладает отличной поддержкой событий и обработки пользовательского ввода, что позволяет создавать отзывчивые и интерактивные приложения.

1.5.3 Работа с файлами

Для удобного хранения баз паролей в приложении используется формат *JSON*, который является текстовым форматом обмена данными. Он позволяет представить информацию в виде удобной для чтения и записи структуры,

состоящей из пар "ключ-значение". Формат *JSON* широко применяется в различных областях программирования и обеспечивает простоту в использовании и интеграцию с другими технологиями.

Для работы с *JSON*-форматами в приложение была интегрирована библиотека «*Newtonsoft.Json».* Эта библиотека является одной из наиболее популярных и мощных библиотек для работы с *JSON* в языке программирования *C#*. Она предоставляет широкий набор функций и возможностей для сериализации (преобразования объектов в *JSON*) и десериализации (преобразования *JSON* в объекты) данных. Библиотека обладает простым и интуитивно понятным *API*, что делает ее использование удобным и эффективным. Она также обладает высокой производительностью и надежностью, что позволяет эффективно работать с большими объемами данных в формате *JSON*.

1.6 Архитектура приложения

В данном приложении используются два основных паттерна проектирования: *MVC* (*Model-View-Controller*). Паттерн *MVC* разделяет приложение на три компонента - модель, представление и контроллер, что обеспечивает разделение ответственностей и упрощает поддержку кода. Графическое представление паттерна можно увидеть на рисунке 1.5.

Дополнительно, в приложении применяются паттерн *Singleton*, который обеспечивает создание единственного экземпляра определенного класса.

Кроме того, при разработке приложения были учтены принципы ООП (объектно-ориентированного программирования). Эти принципы помогают создавать модульный и гибкий код, облегчают его понимание и расширение.

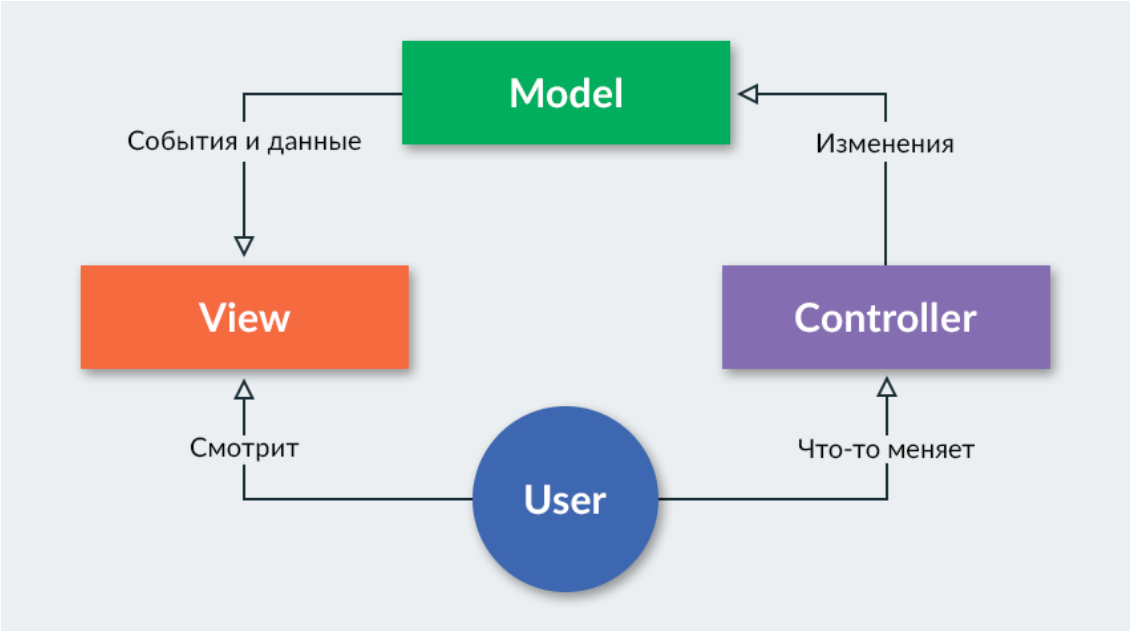


Рисунок 1.5 – Графическое представление *MVC*

1.7 Разработка приложения

1.7.1 Разработка графического интерфейса

Графический интерфейс приложения состоит из различных форм (*Forms*), каждая из которых предназначена для определенного взаимодействия с пользователем. Схема отображения форм представлена на рисунке 1.6.

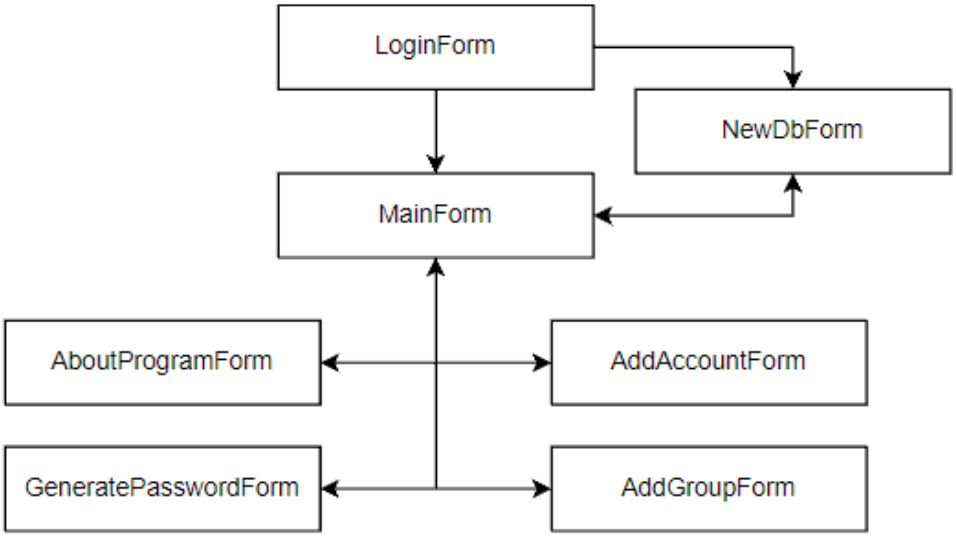


Рисунок 1.6 – Схема отображения форм

Ниже приведены формы, их назначение и функциональность:

- *AboutProgramForm* – форма, отображающая информацию о программе;

- *AddAccountForm* – форма, позволяющая пользователю добавлять новый аккаунт. Она же используется для получения и редактирования информации уже существующего аккаунта;

- *AddGroupForm* – форма, необходимая для возможности пользователя добавлять информацию о новой группе;

- *LoginForm* – форма, предназначенная для входа пользователя в приложение;

- *MainForm* – главная форма приложения, отображающая основной интерфейс и управляющая взаимодействием с другими формами;

- *NewDbForm* – форма, используемая для создания новой базы данных;

- *GeneratePasswordForm* – форма, позволяющая пользователю генерировать пароли.

Каждая форма обеспечивает определенный функционал и предоставляет пользователю возможность взаимодействовать с приложением в соответствии с его задачами и потребностями.

В приложении также используются диалоговые окна (*message box*), предоставляемые встроенными средствами *Windows Forms*. Эти окна позволяют взаимодействовать с пользователем, предоставляя ему информацию, предупреждения или запрашивая подтверждение для определенных действий. Согласно [1], *Windows Form* поддерживает несколько типов необходимых мне диалоговых окон, включая *MessageBox, OpenFileDialog*. Каждое из этих диалоговых окон имеет свою специфику и позволяет взаимодействовать с пользователем в соответствии с требуемыми задачами и сценариями приложения.

1.7.2 Разработка классов

1.7.2.1 Класс *Password*

В классе *Password*, используемом в приложении, определены поля для хранения необходимой информации о пароле. Основные поля класса описаны

ниже:

- *password_masterKey* – поле, предназначенное для хранения мастер-ключа пароля;

- *password_masterKey_len* – поле, указывающее длину мастер-ключа пароля;

- *encryptedPassword* – поле, используемое для хранения зашифрованного пароля;

- *encryptedPassword_len* – поле, указывающее длину зашифрованного пароля;

- *password_salt* – поле, предназначенное для хранения соли пароля;

- *password_salt_len* – поле, указывающее длину соли пароля;

- *password_IV* – поле, используемое для хранения инициализационного вектора пароля;

- *password_IV_len* – поле, указывающее длину инициализационного вектора пароля.

В классе *Password* используются поля, идущие парами для обеспечения защиты пароля в оперативной памяти с помощью метода *ProtectMemory*. *ProtectMemory* может защитить блок оперативной памяти, только если он кратен 16.

Класс также содержит методы и конструкторы для защиты и расшифровки значений пароля, а также функции для копирования пароля в буфер обмена и получения *JSON*-представления пароля. Руководствуясь источником [2], был создан алгоритм шифрования пароля с использованием встроенных библиотек *C#*. При создании экземпляра класса с помощью алгоритма *AES* и ключа пользователя шифруется пароль. Ключ пользователя преобразуется с помощью *Rfc2898DeriveBytes*. Это стандарт формирования ключа на основе пароля. В нашем случае – новый ключ генерируется на основе старого, чтобы пользователь не беспокоился, что заданный им ключ при создании базы паролей был слабым. Класс реализует интерфейс *IDisposable* для освобождения ресурсов и очистки данных пароля при удалении объекта.

На рисунке 1.7 представлены поля и методы класса *Password*.

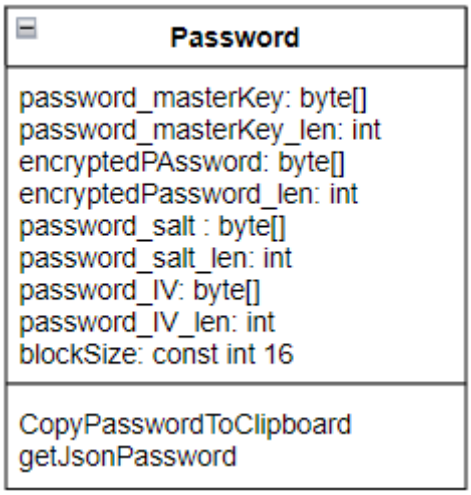


Рисунок 1.7 – Поля и методы класса *Password*

1.7.2.2 Класс *MasterKey*

В классе *MasterKey* определены поля и методы для работы с мастер-ключом пароля. Основные элементы класса описаны ниже:

_key – приватное поле, используемое для хранения мастер-ключа в виде массива байтов;

_keyLen – публичное поле, указывающее длину мастер-ключа;

blockSize – константа, определяющая размер блока (в байтах), используемого для выравнивания длины ключа. Это требуется для защиты ключа в оперативной памяти.

Для защиты мастер-ключа в оперативной памяти используется метод *ProtectedMemory.Protect()*.

Класс *MasterKey* используется в совокупности с классом *Password* для защиты и расшифровки пароля с использованием мастер-ключа.

На рисунке 1.8 представлены поля и методы класса *MasterKey*.

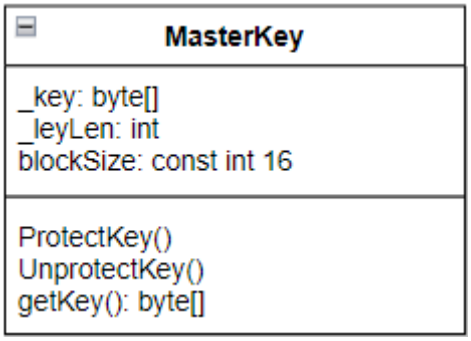


Рисунок 1.8 – Поля и методы класса *MasterKey*

1.7.2.3 Класс *AccountEntry*

Класс *AccountEntry* представляет запись аккаунта и содержит информацию о названии аккаунта, логине, пароле и дополнительных свойствах в виде словаря. Он реализует интерфейс *IDisposable*, что позволяет освобождать ресурсы, используемые объектом.

В классе *AccountEntry* представлены следующие поля:

accountName – строка, содержащая название аккаунта;

login – Строка, содержащая логин аккаунта;

password – объект типа *Password,* который представляет зашифрованный пароль;

properties – словарь, содержащий дополнительные свойства аккаунта;

SerializedPassword – строковое свойство, используемое для сериализации пароля в формат JSON. Внутри свойства используется метод *password.getJsonPassword(),* который возвращает сериализованную строку пароля.

Также в этом классе был реализован индексатор, позволяющий получить или установить значение свойства из словаря *properties*. Если указанный ключ, по которому обратились к объекту, присутствует в словаре Properties, то метод возвращает соответствующее значение, иначе возвращает *null*. При установке значения, оно сохраняется в словаре по указанному ключу.

Класс *AccountEntry* представляет удобную структуру для хранения информации об аккаунте, включая название, логин, пароль и дополнительные

свойства. Он также обеспечивает функциональность для сериализации и десериализации объектов в формат *JSON*.

На рисунке 1.9 представлены поля и методы класса *AccountEntry*.

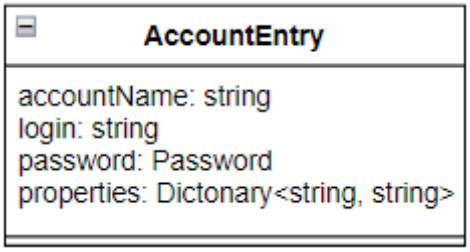


Рисунок 1.9 – Поля и методы класса *AccountEntry*

1.7.2.4 Класс *GroupEntry*

Класс *GroupEntry* представляет запись группы в менеджере паролей. Он содержит следующие основные элементы:

groupList – список объектов *GroupEntry*, представляющих вложенные группы в текущей группе. Этот список используется для хранения и управления подгруппами;

accountEntries – список объектов *AccountEntry*, представляющих учетные записи, связанные с текущей группой. Этот список используется для хранения и управления учетными записями;

name – строка, содержащая имя текущей группы;

Этот класс предоставляет удобное представление информации в древовидной форме, что позволяет организовывать учетные записи и группы в иерархическую структуру. Каждый объект *GroupEntry* представляет отдельную группу и может содержать как подгруппы, так и учетные записи.

На рисунке 1.10 представлены поля и методы класса *GroupEntry*.

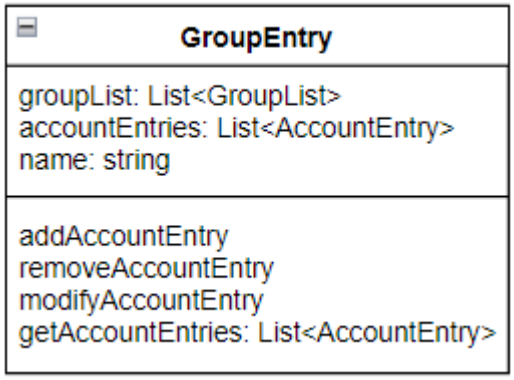


Рисунок 1.10 – Поля и методы класса *GroupEntry*

1.7.2.5 Классы для работы с файлами.

Для работы с файлами были реализованы 2 класса: *FileManager* и *FileEncryptor*. Также для сериализации и десериализации используется *JSON*.

Newtonsoft.Json [3] – это библиотека, которая позволяет контролировать процессы сериализации и десериализации обьектов. Это очень удобно, а также соответствует принципу полиморфизма, так как при изменении классов в приложении обьекты будут записаны в файл, какими бы они ни были. Исключение – класс *Password*, так как для записи его в файл с него сначала необходимо снять защиту *ProtectMemory*.

Из-за этого происходит явное задание поля класса *AccountEntry*:

```
        [JsonIgnore]
        public Password password; //Сначала необходимо исключить это
поле для Json-a
        [JsonProperty("Password")]
        private string SerializedPassword
        {
            get { return password.getJsonPassword(); }
        }
```

В примере кода показано, как это сделать. Сначала исключаем поле, к которому обычно обращаемся в течение программы, а потом создаем приватное поле, которое при вызове «*get*» вызвает метод в классе *Password*:

```
        public string getJsonPassword()
        {
            UnprotectPasswordValuesInOperationalMemory();
            string jPass = JsonConvert.SerializeObject(this);
            ProtectPasswordValuesInOperationalMemory();
            return jPass;
```

}

FileEncryptor используется только в классе *FileManager* и предоставляет функциональность для шифрования и расшифровки *JSON*-объектов и сохранения их в файле. Методы в классе являются статическими, что позволяет вызывать их без создания экземпляра класса.

Класс *FileManager* предоставляет функциональность для загрузки, сохранения и управления файлами базы паролей, а также файлом конфигурации.

Метод *LoadDb* загружает зашифрованный файл по переданному в метод пути и расшифровывает его с использованием переданного объекта *MasterKey*.

Метод *SaveDbToFile* сохраняет корневой объект *GroupEntry* в зашифрованном виде в файл по указанному пути.

Также в классе реализованы методы для загрузки и сохранения файла конфигурации.

Класс *FileManager* так же содержит статические методы, поэтому можно использовать их без создания экземпляра класса.

FileManager использует стандартные методы работы с файлами, которые также описаны в источнике [4].

На рисунке 1.11 представлены поля и методы классов *FileManager* и *FileEncryptor*.

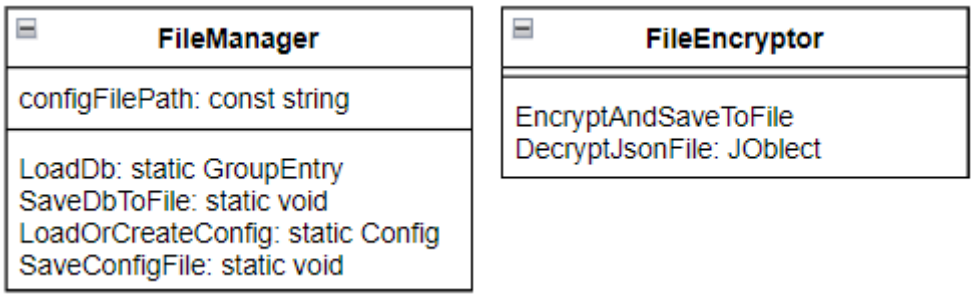


Рисунок 1.11 – Поля и методы классов *FileManager* и *FileEncryptor*

1.7.2.6 Класс *MainForm*.

Класс *MainForm* представляет главную форму приложения. В этом классе размещается пользовательский интерфейс, логика взаимодействия с пользователем и управление другими классами и компонентами.

В классе *MainForm* реализованы два встроенных класса: *TreeViewManager* и *ListViewManager*.

Класс *TreeViewManager* отвечает за управление элементом *TreeView* на форме. Он обеспечивает функциональность добавления, удаления и обновления узлов дерева, а также обработку событий, связанных с выбором и раскрытием узлов.

Объект *ListViewManager* отвечает за управление элементом управления *ListView* на форме. Он обеспечивает функциональность отображения списка аккаунтов, добавления, удаления и обновления элементов списка, а также обработку событий, связанных с выбором элементов списка.

Класс *MainForm* содержит поля *MasterKey*, *Config* и *GroupEntry*, которые играют важную роль в управлении безопасностью, настройками и структурой данных приложения. *MasterKey* используется для шифрования и расшифровки ключа доступа, *Config* хранит конфигурационные данные приложения, а *GroupEntry* предоставляет удобное представление информации о группах и аккаунтах.

1.7.2.7 Отношения между объектами классов

Отношение между объектами классов моделей показаны на рисунке 1.12. Также на этом рисунке указана кратность, в соответствии с источником [5]. Кратность – это количество экземпляров одного класса которые могут быть с связаны с экземпляром другого класса. Значение кратности указывается в виде диапазона «1» (ровно один), «0..*» (ноль и более).

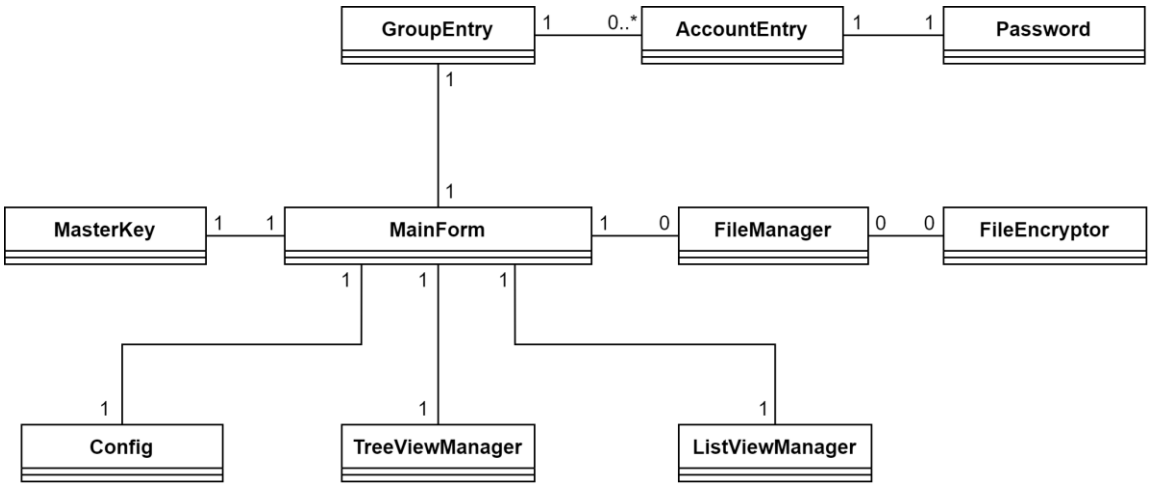


Рисунок 1.12 – *UML*-диаграмма объектов

Класс *MainForm* связан к классом *GroupEntry* отношением один ко многим. В *MainForm* содержится одна корневая запись *GroupEntry*.

Класс *GroupEntry* связан с *AccountEntry* отношением один ко многим. В одной группе может быть множество записей аккаунтов. Также может быть ни одной записи аккаунта.

Класс *AccountEntry* связан с *Password* один к одному. У одного аккаунта может быть только один пароль.

Класс *MainForm* связан с классом *MasterKey* один к одному. В *MainForm* всегда открыта только одна база паролей, а у одной базы паролей может быть только один мастер-ключ.

Класс *MainForm* связан с классом *FileManager* один к нулю. *FileManager* обладает статическими методами, для которых необязательно создавать объект класса.

Класс *FileManager* связан с классом *FileEncryptor* ноль к нулю. *FileEncryptor*, как и *FileManager* обладает статическими методами и не имеет полей, ему не обязательно создавать объект класса для выполнения методов.

Класс *MainForm* связан с классом *TreeViewManager* один к одному. *TreeViewManager* в одном экземпляре состоит в классе *MainForm*.

Класс *MainForm* связан с классом *ListViewManager* один к одному. *ListViewManager* в одном экземпляре состоит в классе *MainForm*.

Класс *MainForm* связан с классом *Config* один к одному. Конфигурация загружается единожды и в одном экземпляре.

1.8 Описание приложения

1.8.1 Руководство пользователя

При первом запуске приложения потребуется выбрать директорию, в которую будут сохраняться ваши базы паролей. Откроется стандартная форма выбора директории.

После выбора директории откроется окно входа в приложение (рисунок 1.13).

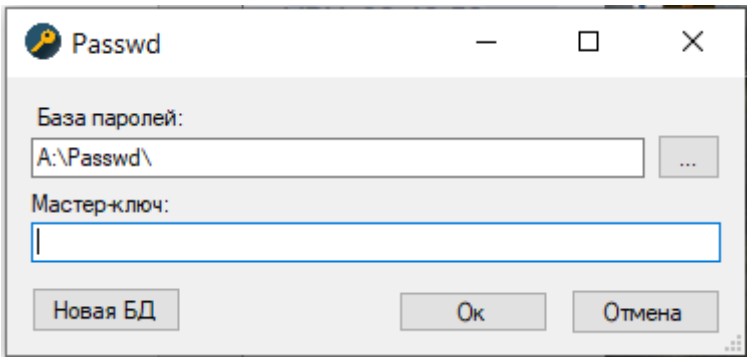


Рисунок 1.13 – Форма входа в приложение

Так как базы паролей пользователя еще не существует, необходимо создать новую БД. При нажатии на кнопку «Новая БД» откроется форма ввода данных для создания новой базы паролей (рисунок 1.14).

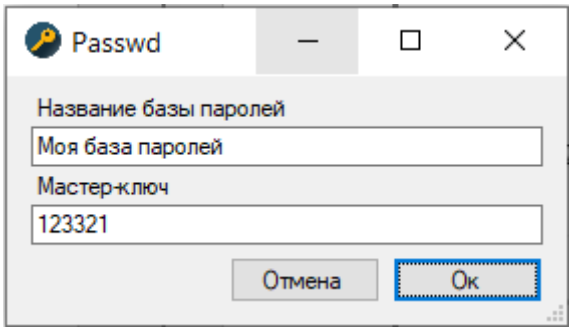


Рисунок 1.14 – Создание новой базы паролей

При нажатии кнопки «Отмена» приложение закрывается. При вводе данных, и нажатии кнопки «Ок», откроется главная форма редактирования. (рисунок 1.15).

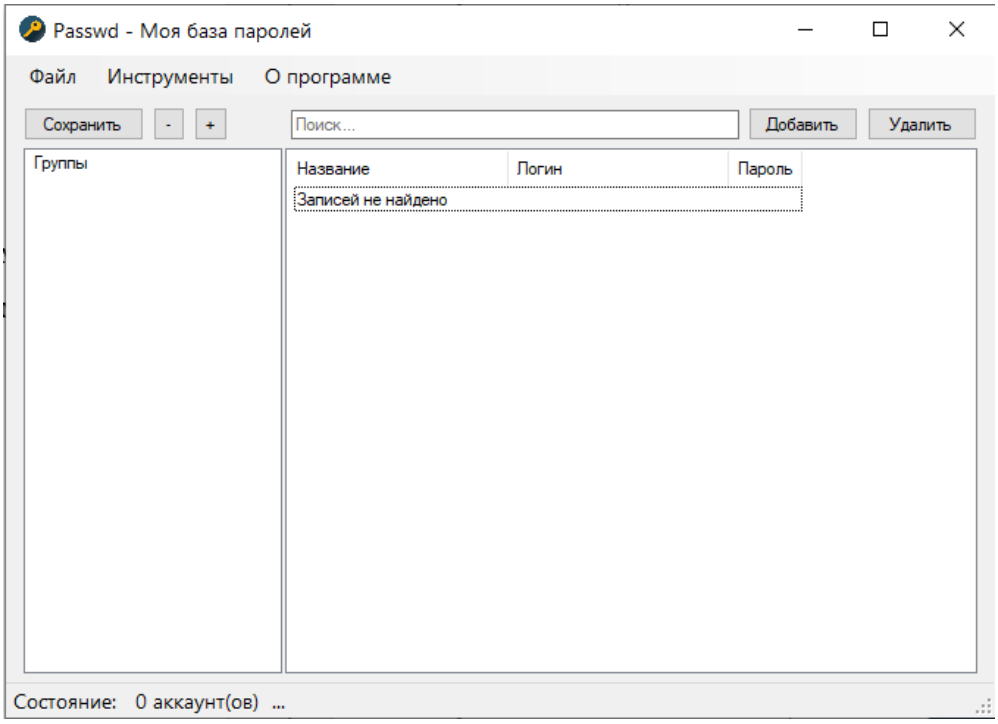


Рисунок 1.15 – Главная форма редактирования базы паролей

В левой части формы находится управление группами. При создании базы паролей всегда создается корневая директория под названием «Группы», которую невозможно удалить. Кнопка «+» отвечает за создание новой группы, кнопка «-» – за удаление группы. При удалении группы также удаляются все аккаунты, которые входили в эту группу, а также вложенные группы. Кнопка «Сохранить» отвечает за сохранение базы паролей в файл.

В правой части формы находится управление аккаунтами в выбранной группе. При создании новой базы паролей по умолчанию выбранной группой считается корневая. Можно добавлять пароли прямо в неё.

При нажатии кнопки «+» откроется форма создания новой группы (рисунок 1.16).

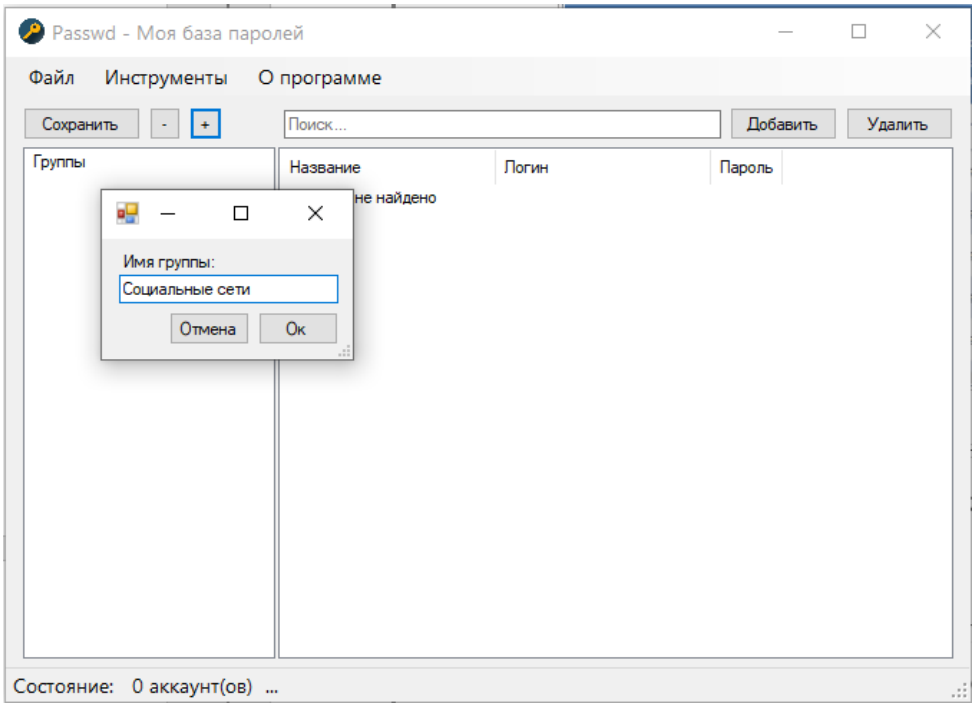


Рисунок 1.16 – Создание группы

При нажатии «Отмена» создание группы отменится. Группа не будет создана. При нажатии «Ок» создастся подгруппа в выбранной заранее группе. Если группа, внутри которой необходимо создать новую, не была выбрана, то группа создастся в корне. На рисунке 1.17 показана программа, в которой были созданы несколько групп.

При создании нескольких групп пользователь может переключаться между ними левым щелчком мыши. Для добавления аккаунта в группу следует выбрать необходимую группу, а затем нажать кнопку «Добавить».

При добавлении аккаунта появляется форма (рисунок 1.18), которая предлагает заполнить данные аккаунта. При нажатии кнопки «Отмена» форма закроется и создание аккаунта будет отменено. При нажатии кнопки «+» добавится дополнительное поле, которое может понадобиться для прочей информации, например, ответ на секретный вопрос, или номер телефона, привязанный к аккаунту. При задержании клика на кнопке с глазом можно посмотреть замаскированный введённый пароль. При нажатии кнопки «Ок» введённая информация сохранится и аккаунт будет создан.

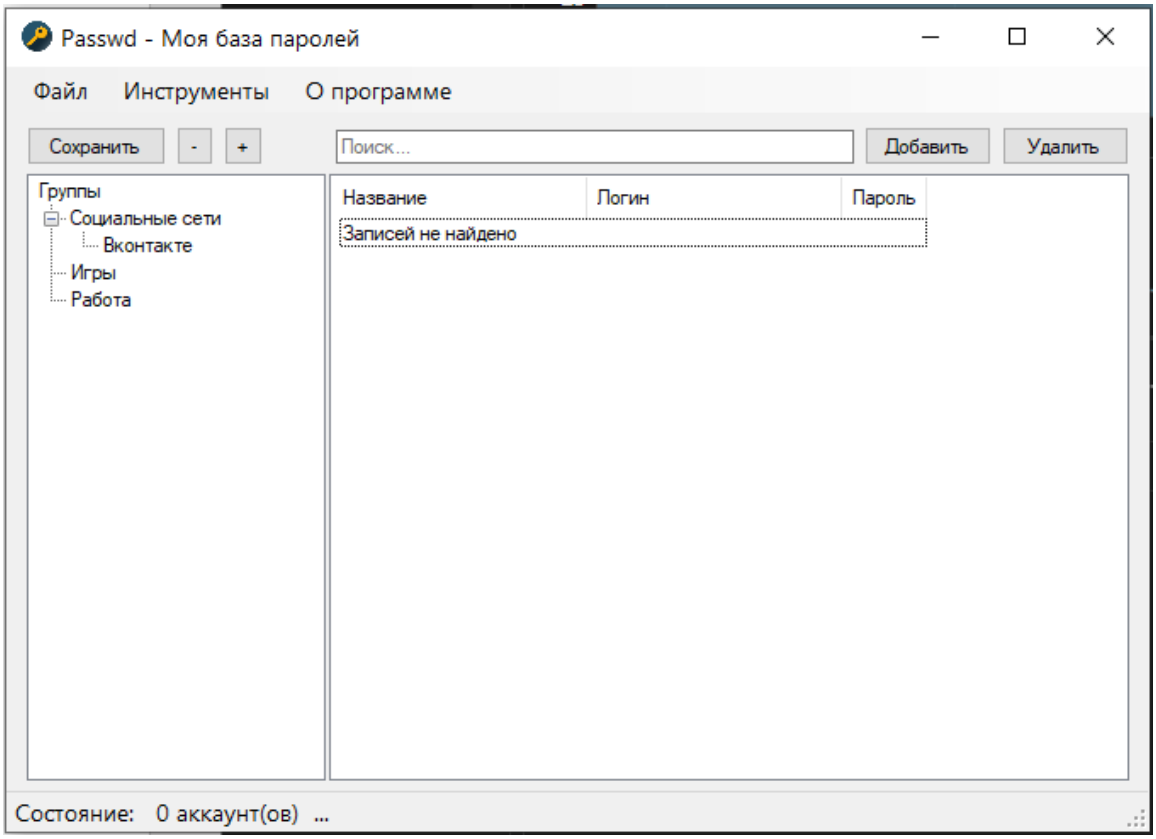


Рисунок 1.17 – Созданные пользователем группы

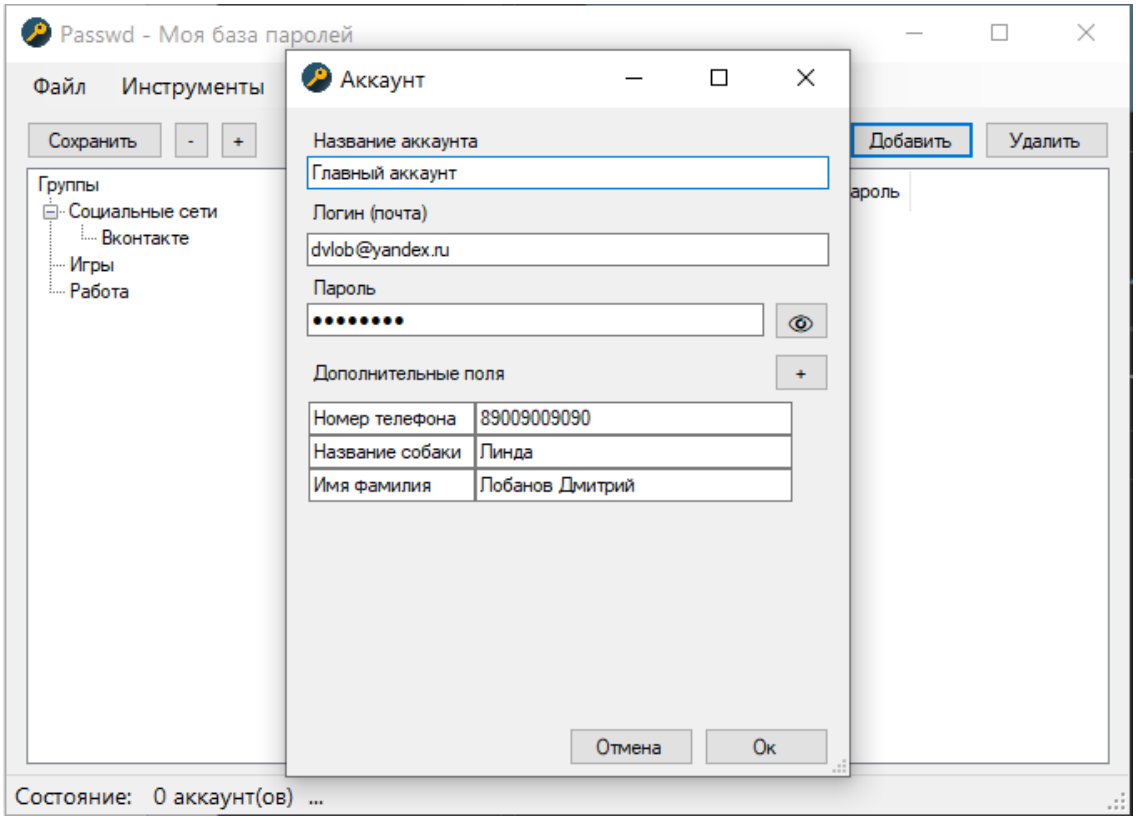


Рисунок 1.18 –Добавление нового аккаунта

На рисунке 1.19 показана заполненная данными база паролей. Теперь после добавления аккаунта по нему можно сделать двойной щелчок мыши. Пароль, зашифрованный в записи, расшифруется и будет помещен в буфер обмена на 10 секунд. 10 секунд является достаточным временным интервалом, чтобы вставить скопированный пароль в поле, где он необходим.

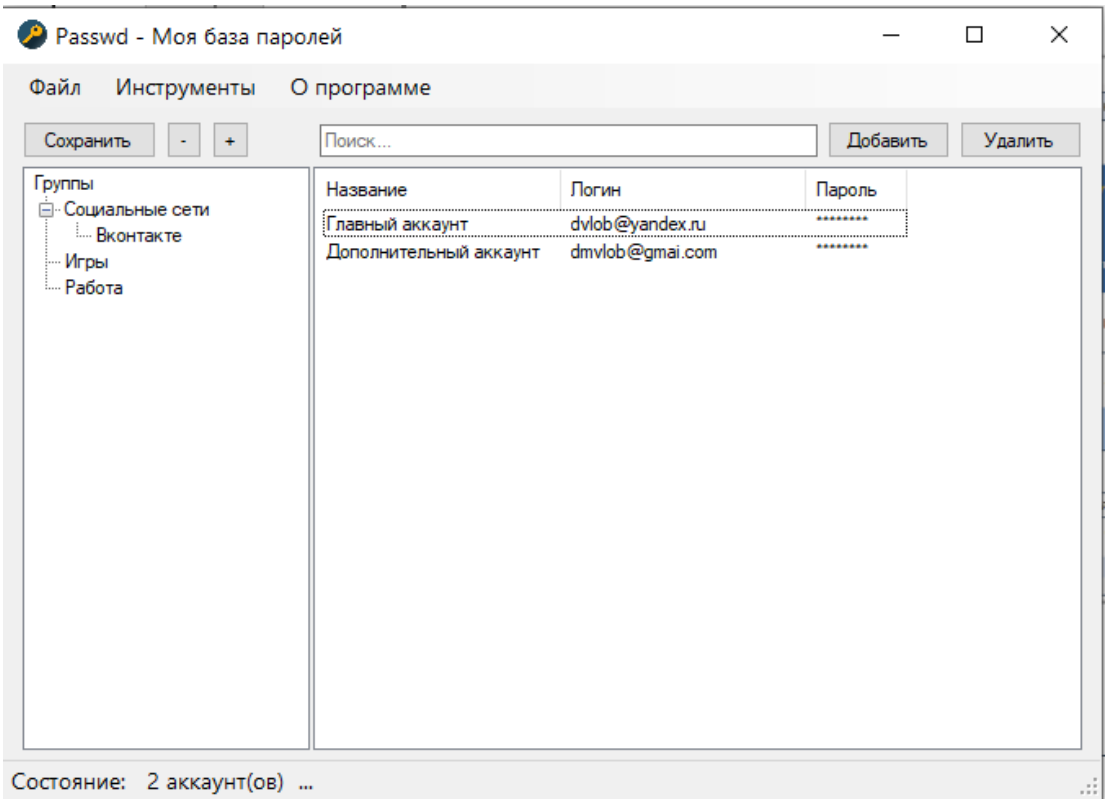


Рисунок 1.19 –Заполненная база паролей

При нажатии правой кнопкой мыши по аккаунту откроется вся информация об аккаунте (рисунок 1.20), за исключением пароля. Вместо пароля будет надпись: «Измените это поле, чтобы поменять пароль». В этой форме можно спокойно менять все данные, и при нажатии кнопки «Ок» все измененные данные сохранятся. Если сохранять изменения нет необходимости, можно нажать кнопку «Отмена».

При выходе из приложения будет предложено сохранить изменения базы паролей. Можно отменить все изменения, и, если пользователь не сохранил новую созданную базу паролей, файла с сохраненной базой паролей не будет.

Если пользователь открыл уже имеющуюся базу паролей для изменения, и закрыл без сохранения изменений, то в файле будет сохранена версия базы паролей, которая была до открытия её в приложении.

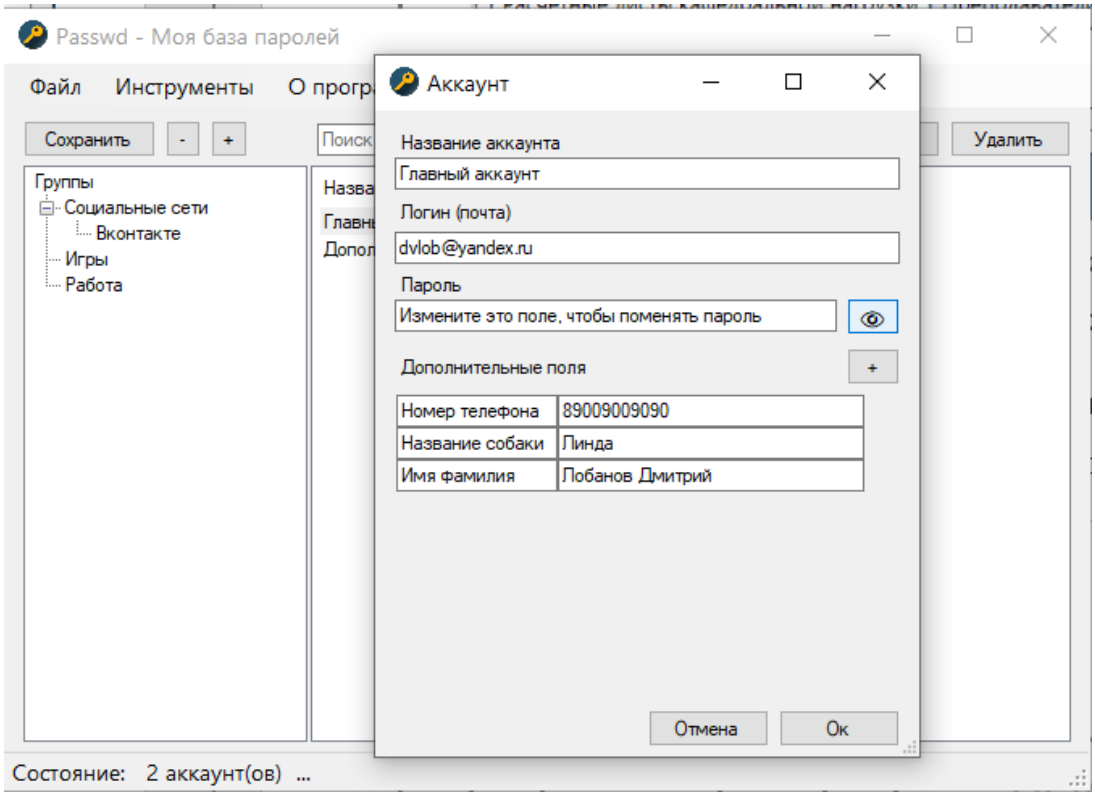


Рисунок 1.20 – Полная информация об аккаунте

При повторном запуске приложения, ваша база паролей уже будет введена в файл для открытия, так как при сохранении базы паролей также меняется файл конфигурации, в котором запоминается последняя открытая база паролей.

В форме входа также можно создать новую базу, или выбрать файл уже существующей базы паролей, нажав кнопку «…» (Рисунок 1.21).

При выборе определенного файла в форме входа будет указан путь до файла с его именем, чтобы пользователь полностью мог удостовериться, что он открывает нужный ему файл. При вводе мастер-ключа можно нажать «*Enter*» и произведется попытка открытия базы паролей.

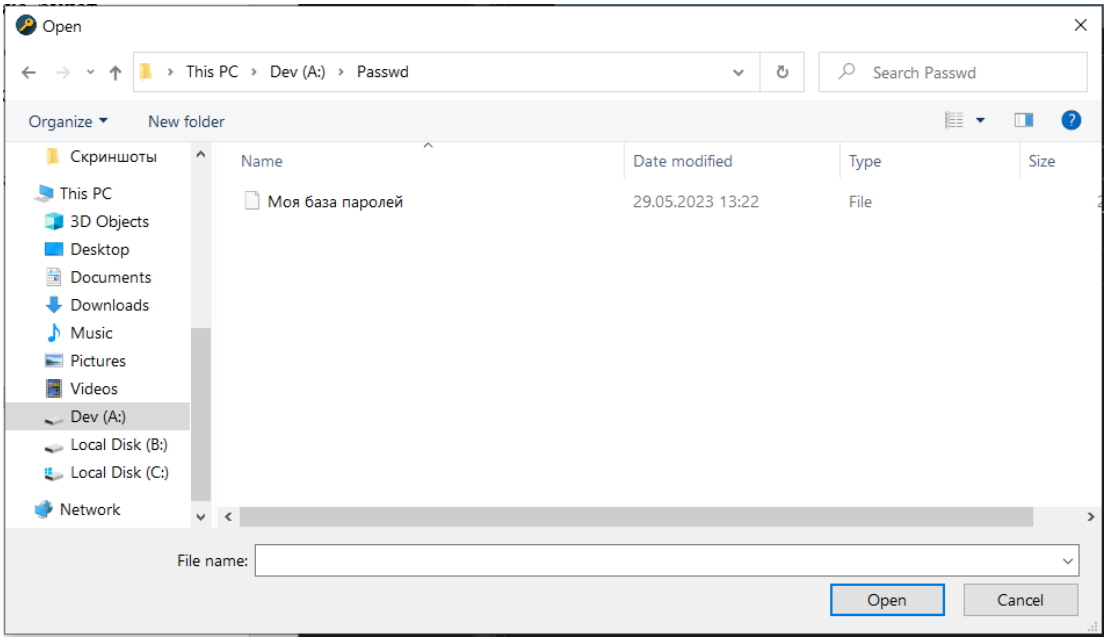


Рисунок 1.21 –Выбор файла базы паролей при входе

Если мастер-ключ неверен, форма входа закроется и появится предложение о создании новой базы паролей (рисунок 1.22).

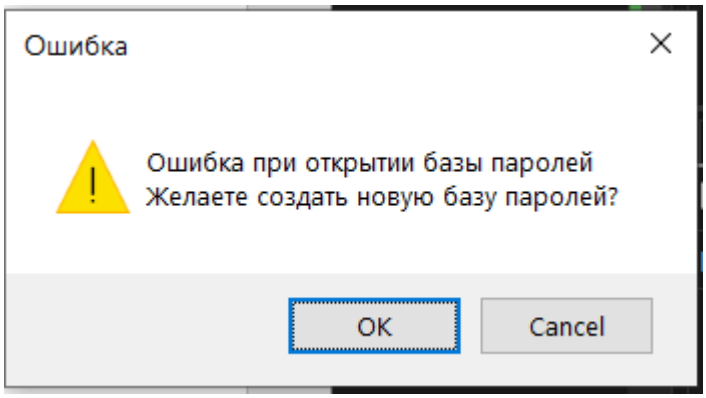


Рисунок 1.22 – Неверный мастер-ключ

При нажатии «*OK*» откроется форма создания новой базы паролей. При нажатии «*Cancel*» приложение закроется.

В панели инструментов, при выборе поля «Генерация пароля», откроется инструмент автоматической генерации пароля (рисунок 1.23).

Рисунок 1.23 – Инструмент генерации пароля

В этой вкладке можно выбрать, какой тип символов необходим в пароле, поставить галочки около соответствующих символов, задать длину, и нажать «Генерировать». При нажатии кнопки в текстовом поле под надписью «Пароль» появится сгенерированный пароль, пригодный в дальнейшем использовании.

2 Охрана труда

2.1 Анализ негативных факторов, воздействующих на оператора ПЭВМ и способы их устранения

Работа с вычислительной техникой происходит в условиях ограниченной подвижности и длительного статического мышечного напряжения, что может приводить к негативным последствиям, таким как снижение работоспособности, переутомление, психологическое раздражение, физические недомогания и общая усталость. Для снижения негативного воздействия на человека, важно придерживаться определенных правил использования компьютера.

Основными, из которых являются:

– режим труда и отдыха;

– правильная организация рабочего места;

– правила электробезопасности;

– правила пожарной безопасности.

2.1.1 Оборудование рабочего места

Ниже представлены основные требования, которым должно удовлетворять рабочее место:

– расстояние от глаз до экрана должно быть не менее 500 мм;

– монитор должен располагаться ниже уровня глаз.

Угол между направлением при взгляде прямо и направлением на центр экрана должен составлять примерно 30 градусов;

– высота рабочего стола должна составлять от 620 до 820 мм;

– рабочий стол должен иметь пространство для ног высотой не менее 600 мм, глубиной на уровне колен – не менее 450 мм и на уровне вытянутых ног – не менее 650 мм.

На рисунке 2.1 изображена организация рабочего места, которая отвечает указанным требованиям.

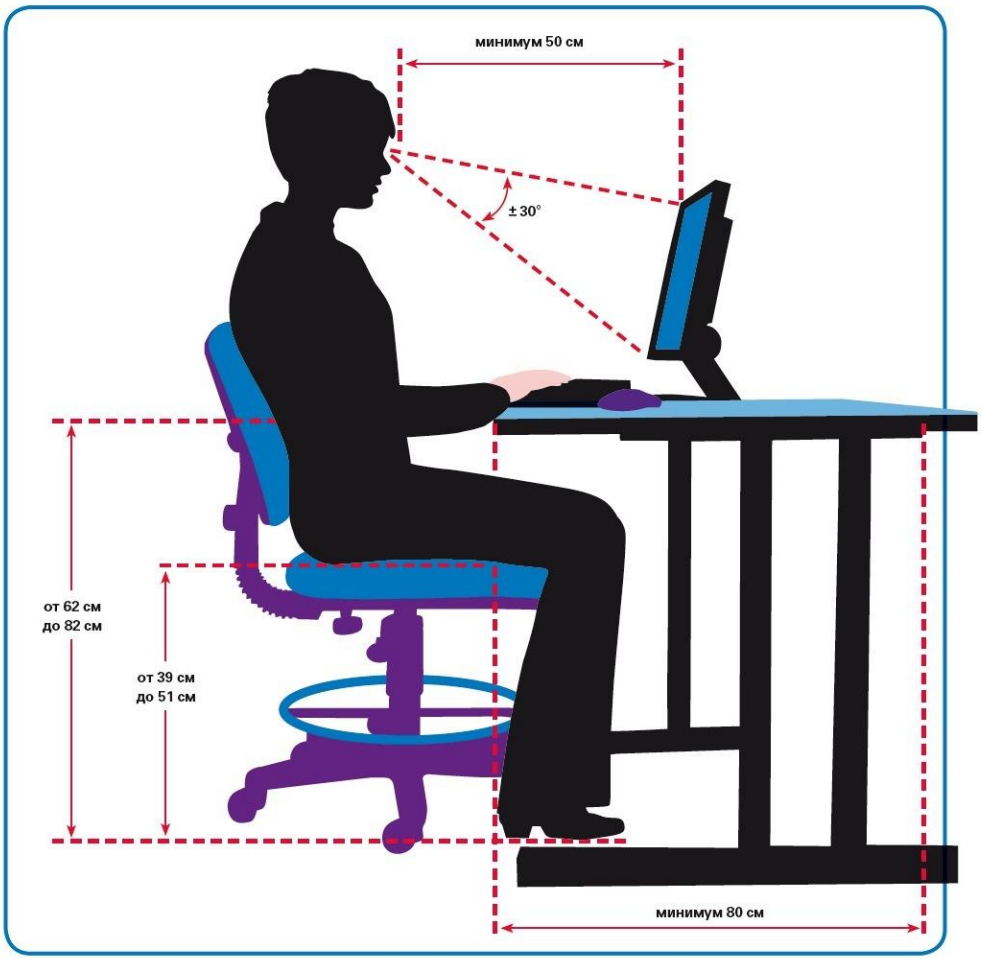


Рисунок 2.1 – Организация рабочего места

Рабочее место с дисплеем должно обеспечивать оператору возможность удобного выполнения работ в положении сидя и не создавать перегрузки костно-мышечной системы.

При правильно организованном рабочем месте должна обеспечиваться правильная посадка пользователя ПК, изображённая на рисунке 2.2

Посадка должна обеспечивать выполнение следующих условий:

– запястья должны находиться в прямом положении;

– плечи находятся в расслабленном состоянии;

– локти должны не свисать и располагаться перпендикулярно полу (угол в локтевых суставах должен составлять 90-100 градусов).

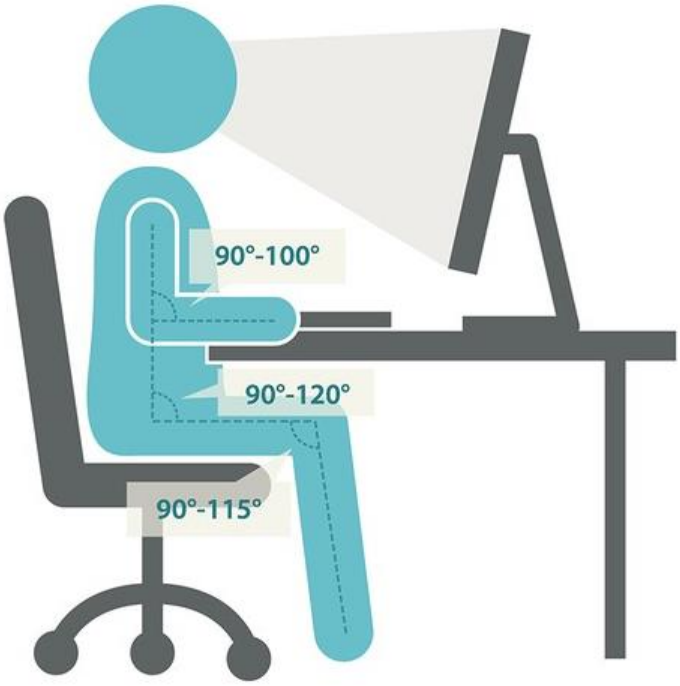


Рисунок 2.2 – Правильная посадка за ПК

2.1.2 Освещенность рабочего места

Для обеспечения комфортных условий работы на компьютере важно учитывать стандарты освещения, которые зависят от характера работы, расположения и поверхности рабочего места. Рекомендуется достигать комбинированного освещения клавиатур и дисплеев не менее 400 люкс и общего освещения не менее 200 люкс. Одновременно необходимо обеспечивать равномерную яркость на рабочем месте, избегая ярких и блестящих предметов. Для этого часто используются потолочные или встроенные светильники с люминесцентными лампами, которые рекомендуется располагать сбоку от рабочего места, параллельно линии зрения пользователя, чтобы предотвратить засветку экранов.

2.1.3 Электробезопасность

К электробезопасности предъявляются требования по ГОСТ 12.1.019-2017 «ЭЛЕКТРОБЕЗОПАСНОСТЬ».

ПК – это комплекс устройств, работающий от сети переменного тока частотой 50 Гц с напряжением 230 В. Электрический ток при таком напряжении опасен для жизни. Поэтому при работе на ПК необходимо соблюдать следующие требования электробезопасности:

— все устройства должны питаться от одной фазы электрической сети;

— сетевое электропитание устройств ПК должно производиться только от розеток типа «Европа» с заземляющими контактами;

— все электрические розетки, предназначенные для подключения к ним устройств ПК, должны иметь маркировку по напряжению;

— значение номинального напряжения сети необходимо наносить яркой краской, крупными символами (высотой не менее 50 мм) на стене или щите, возле розетки.

В процессе использования компьютерной техники и периферийных устройств каждый сотрудник должен соблюдать меры безопасности и быть внимательным к электропроводке, приборам и оборудованию, так как нарушение правил безопасности может представлять угрозу для здоровья и жизни. Необходимо регулярно проверять состояние электропроводки, выключателей и розеток, которые используются для подключения оборудования к сети. Если обнаружена неисправность, необходимо немедленно отключить электрооборудование. Продолжение работы возможно только после устранения всех неисправностей.

2.1.4 Пожарная безопасность

При эксплуатации электроустановок и ПК наиболее частыми причинами пожаров, возникающих при использовании электроустановок, являются:

– короткие замыкания в проводках и электрических устройствах;

– воспламенение горючих материалов вблизи от электрических устройств;

– большие переходные сопротивления в местах контактных соединений.

Пожарные меры профилактики проводятся согласно федеральному закону от 21.12.1994 №69-ФЗ (ред. от 28.09.2010) "О пожарной безопасности". Для

пожарной безопасности требуется выполнить ряд организационных мероприятий, направленных на обеспечение безопасности людей и предотвращение пожара с ограничением его распространения. А также создать условия для успешного тушения пожаров. Для повышения противопожарной безопасности помещения необходимы следующие мероприятия:

– проходы в помещения и запасные выходы не должны быть загромождены посторонними предметами, которые препятствуют свободному прохождению;

– к средствам пожаротушения всегда должен быть свободный доступ;

– система вентиляции должна быть оборудована устройствам, обеспечивающим автоматическое отключение ее при пожаре, а также огне задерживающим клапаном;

– установленная автоматическая пожарная сигнализация и средства пожаротушения должны содержаться в исправном состоянии и в соответствии с типовыми правилами технического содержания установок пожарной автоматики.

2.1.5 Режим труда и отдыха

Работник, работающий с ПК, обязан соблюдать режим труда и отдыха в зависимости от продолжительности, вида и категории трудовой деятельности.

Существуют следующие виды трудовой деятельности, связанные с персональным компьютером:

– группа А – работа по считыванию информации с экрана видеодисплейного терминала с предварительным запросом;

– группа Б – работа по вводу информации;

– группа В – творческая работа в режиме диалога с ПЭВМ.

При выполнении работ, относящихся к различным видам трудовой деятельности, за основную работу с компьютером следует принимать тот вид деятельности, который занимает не менее 50% времени в течение всей рабочей смены или рабочего дня.

Для видов трудовой деятельности устанавливается 3 категории тяжести и напряженности работы с компьютером, которые определяются следующим образом:

– для группы А – по суммарному числу считываемых знаков за рабочую смену (не более 60000 знаков за смену);

– для группы Б – по суммарному числу считываемых или вводимых знаков за рабочую смену (не более 40000 знаков за смену);

– для группы В – по суммарному времени непосредственной работы с компьютером за рабочую смену (не более 6 часов за смену).

Суммарное время перерывов для каждой категории и видов работы указаны в таблице 2.1.

Таблица 2.1 – Время перерывов

Категория работы с ПЭВМ или ВДТ	Уровень нагрузки за рабочую смену при видах работы с ВДТ			Суммарное время регламентированных перерывов, мин	
	группа А, количество знаков	группа Б, количество знаков	группа В час	при 8-часовой смене	при 12-часовой смене
1	До 20000	До 15000	До 2	50	80
2	До 40000	До 30000	До 4	70	110
3	До 60000	До 40000	До 6	90	140

Заключение

В процессе разработки менеджера паролей на основе *Windows Forms* была создана автоматизированная система для безопасного хранения и управления паролями. Приложение предоставляет пользователю возможность импортировать и хранить информацию о пользовательских учетных записях, организовывать пароли по категориям и выполнять поиск и фильтрацию для удобного доступа. Оно также обеспечивает безопасное шифрование и хранение паролей, а также предлагает функции автоматической генерации надежных паролей. В будущем можно рассмотреть улучшения, такие как облачная синхронизация, улучшение интерфейса пользователя с использованием дополнительных элементов управления и функциональности, повышение стабильности и отказоустойчивости, а также возможность напоминания об истечении срока пароля.

Данная работа открывает перспективы для импортирования разработанной технологии менеджера паролей на мобильные устройства, такие как смартфоны и планшеты. Это позволит пользователям удобно и безопасно управлять своими паролями в любое время и в любом месте. Приложение для мобильных устройств может предоставлять аналогичные функции хранения, генерации и управления паролями, а также синхронизировать данные с версией для настольных компьютеров. Это значительно повысит удобство использования и доступность менеджера паролей для широкого круга пользователей и улучшит общую безопасность и защиту их учетных записей.

Список использованных источников

1. *MSDN Microsof*t [Электронный ресурс]. *URL*: *https://learn.microsoft.com/ru-ru/dotnet/csharp/* (дата обращения: 08.05.2023).
2. *Applied Cryptography*: *Protocols, Algorithms and Source Code in C* / Брюс Шнайер — Издательство Вильямс, 2016
3. Библиотека *Newtonsoft.Json* [Электронный ресурс]. *URL: https://www.newtonsoft.com/json* (дата обращения: 15.05.2023).
4. Язык программирования *C#* 7 и платформы *.NET и .NET Core*, 8-е изд.: Пер. с англ. — СПб.: ООО “Диалектика”.
5. Дж. Рамбо, М. Блаха *UML* 2.0. Объектно-ориентированное моделирование и разработка, 2–е издание. – СПБ.: Питер, 2007

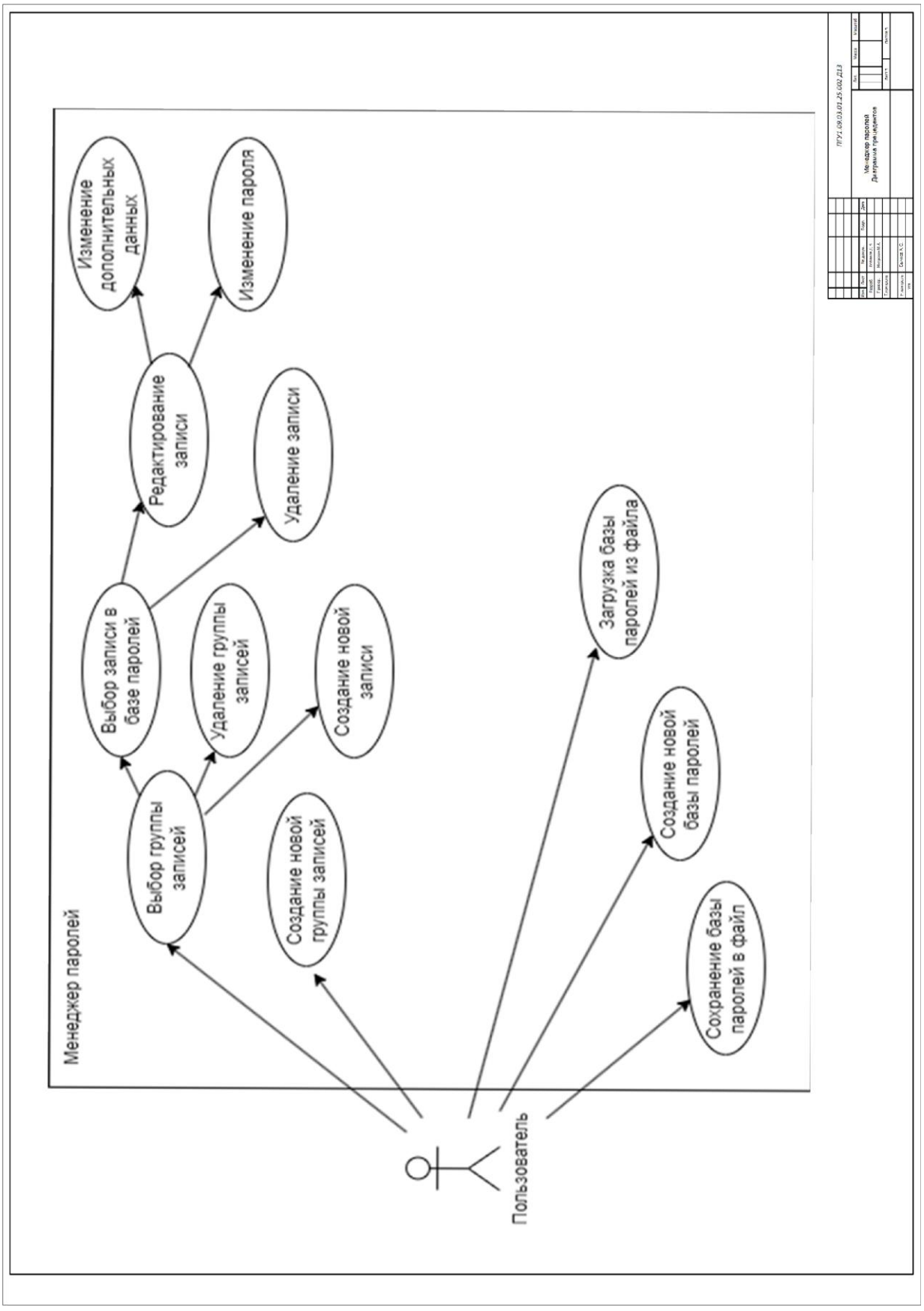
Приложение А

(Обязательное)

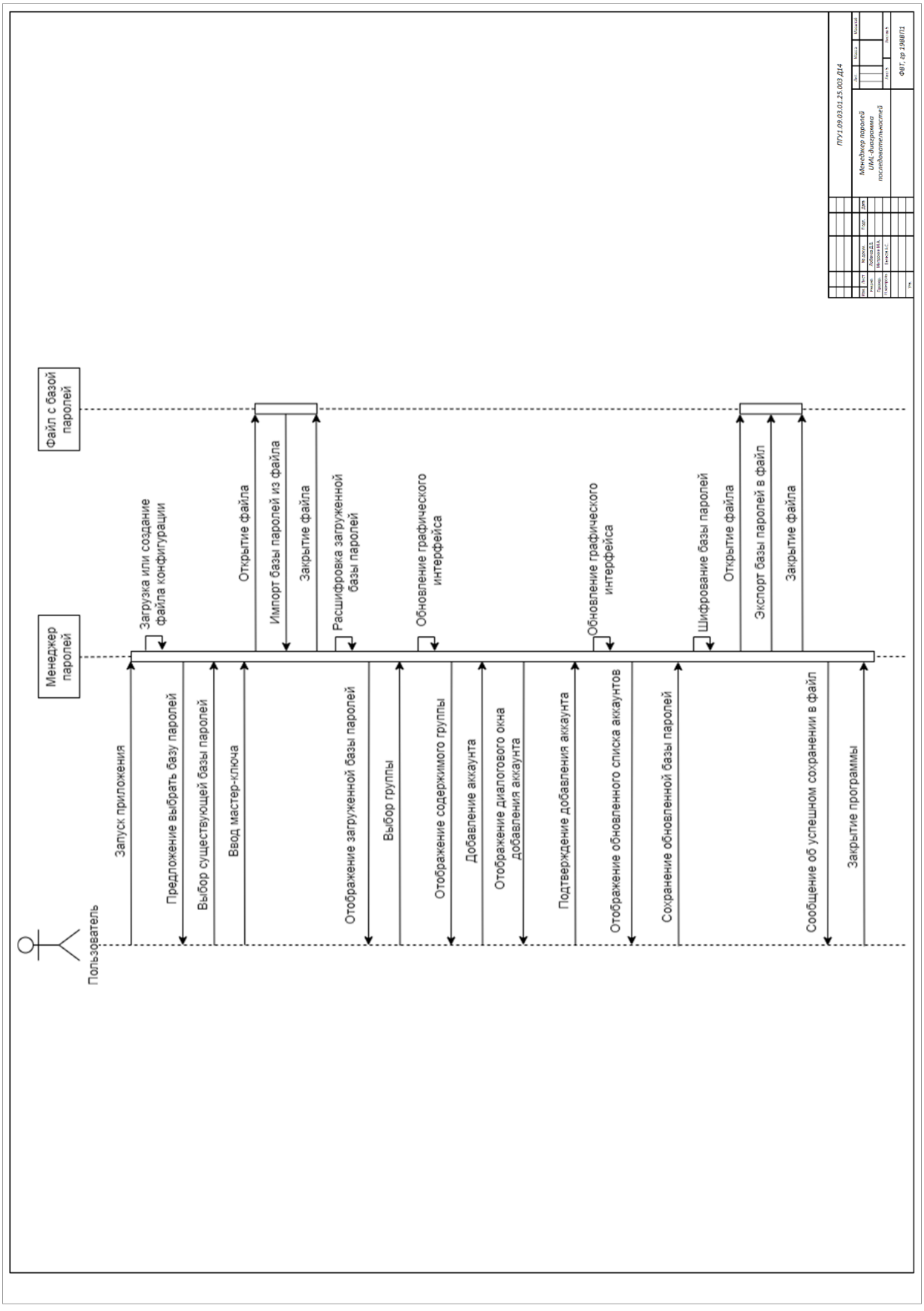
Менеджер паролей

UML-диаграммы

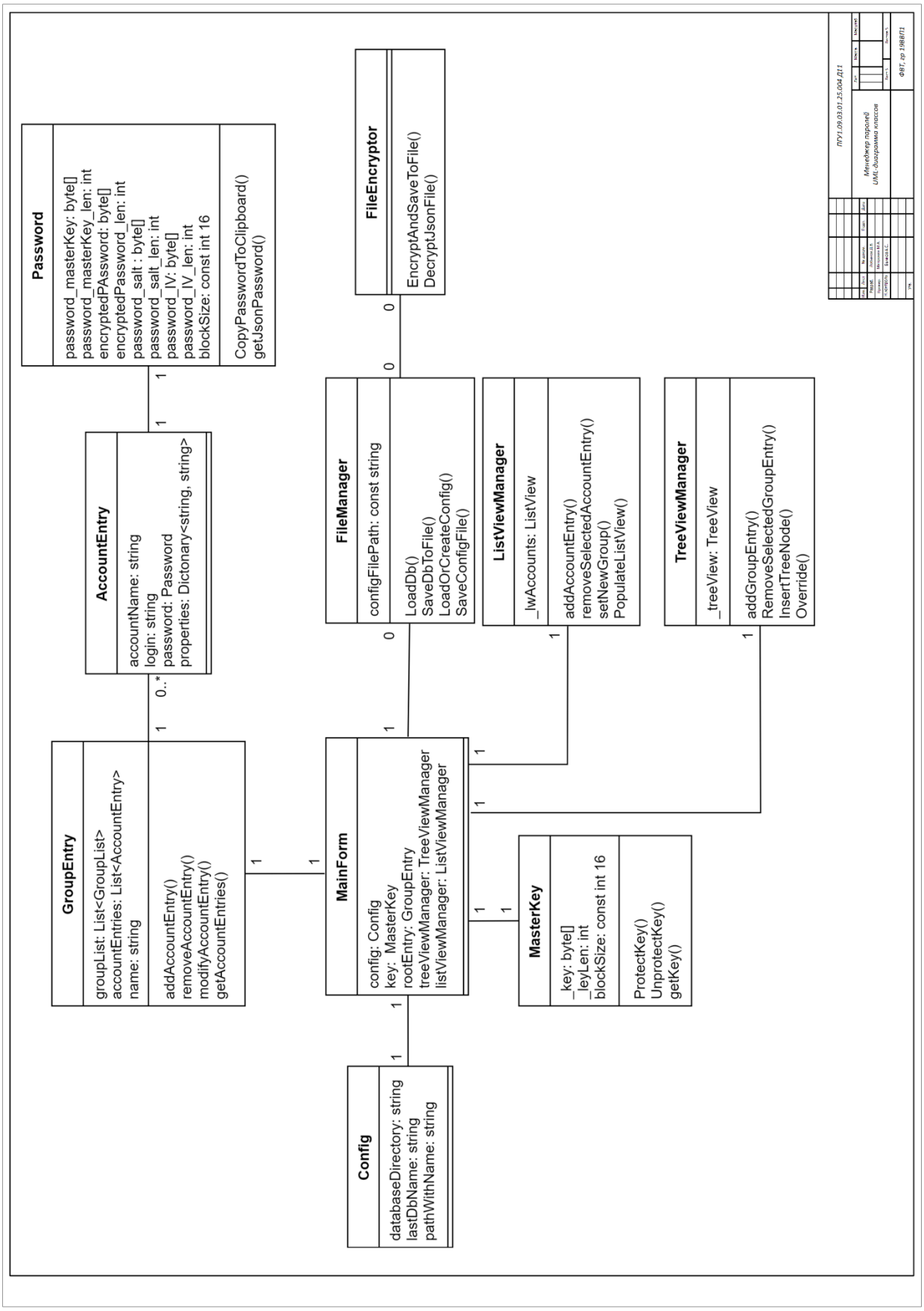
Приложение А.1 – UML-диаграмма прецедентов



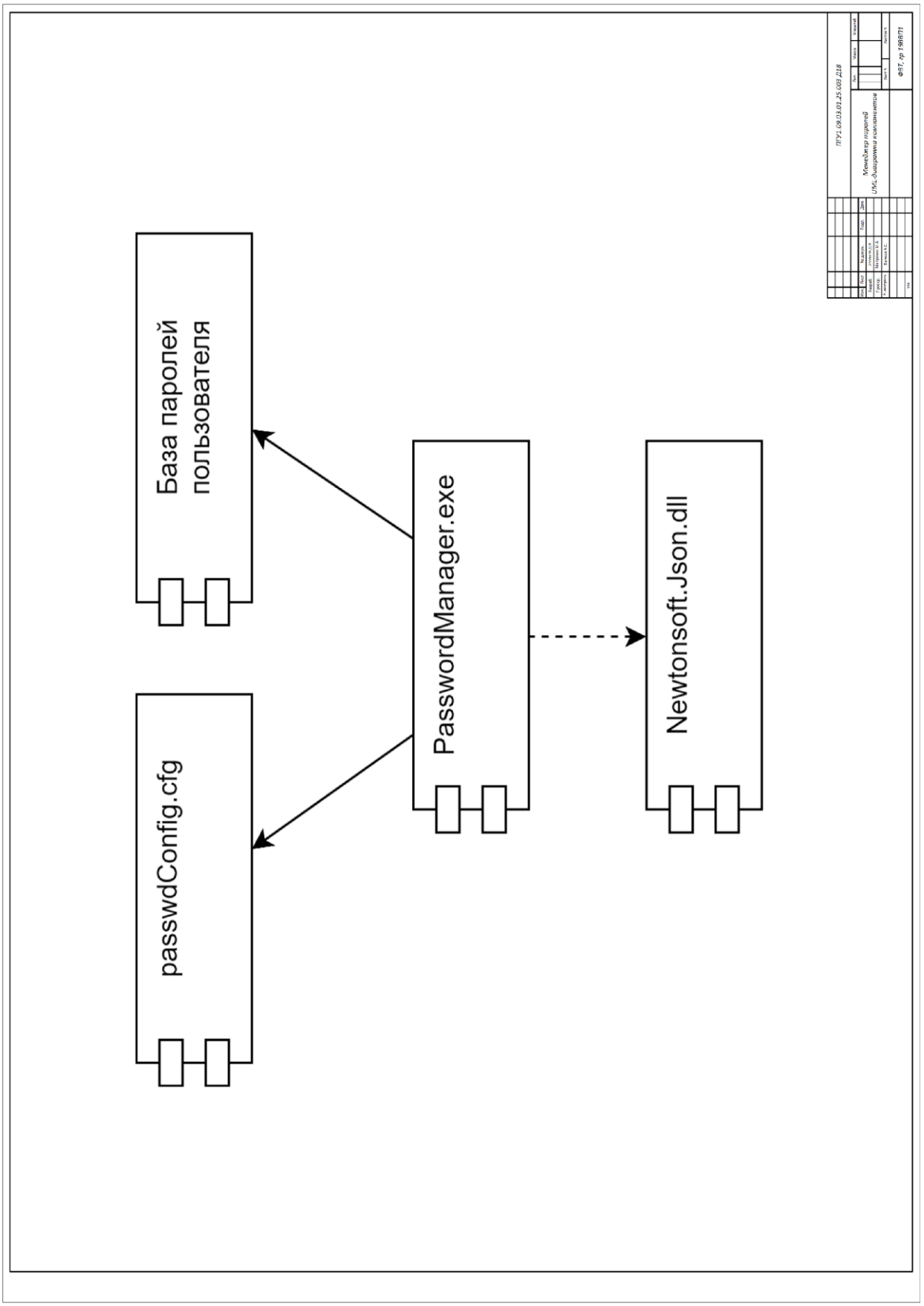
Приложение А.2 – UML-диаграмма последовательностей



Приложение А.3 – UML-диаграмма классов



Приложение А.4 – UML-диаграмма компонентов



Приложение Б

(Обязательное)

Менеджер паролей

Листинг программы

Файл «*Password.cs*»

```
using System;
using System.Text;
using System.Security.Cryptography;
using System.Windows.Forms;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

using System.Timers;
using Timer = System.Timers.Timer;
using System.Security.Cryptography.Xml;
using PasswordManager.Classes;
using System.Threading;

namespace PasswordManager
{
    public class Password : IDisposable
    {
        //отдельно указывать длину каждого поля необходимо для
функции Protect и unprotect (тк в оперативке оказывается защита
памяти идет блоками по 16 байт)
        [JsonProperty]
        byte[] password_masterKey;
        [JsonProperty]
        int password_masterKey_len;
        [JsonProperty]
        byte[] encryptedPassword;
        [JsonProperty]
        int encryptedPassword_len;
        [JsonProperty]
        byte[] password_salt;
        [JsonProperty]
        int password_salt_len;
        [JsonProperty]
        byte[] password_IV;
        [JsonProperty]
        int password_IV_len;
        [JsonProperty]
        const int blockSize = 16;
        public Password(string password, MasterKey key)
        {
            setPassword(password, key);
        }
        ///<Summary>
        ///Конструктор класса для Json
        ///</Summary>
        [JsonConstructor]
        private Password(byte[] password_masterKey,int
password_masterKey_len,
            byte[] encryptedPassword,             int
encryptedPassword_len,
```

```
            byte[] password_salt,                   int
password_salt_len,
            byte[] password_IV,                     int
password_IV_len)
        {
            this.password_masterKey = password_masterKey;
            this.password_masterKey_len = password_masterKey_len;
            this.encryptedPassword = encryptedPassword;
            this.encryptedPassword_len = encryptedPassword_len;
            this.password_salt = password_salt;
            this.password_salt_len = password_salt_len;
            this.password_IV = password_IV;
            this.password_IV_len = password_IV_len;
            ProtectPasswordValuesInOperationalMemory();
        }

        private byte[] AlignToBlockSize(byte[] data)
        {
            int paddingSize = blockSize - (data.Length % blockSize);
            byte[] alignedData = new byte[data.Length +
paddingSize];
            Array.Copy(data, alignedData, data.Length);
            return alignedData;
        }
        private byte[] RestoreOriginalLength(byte[] data, int
originalLength)
        {
            if (data.Length == originalLength)
            {
                return data; // Если длина уже соответствует
исходной, возвращаем массив без изменений
            }

            byte[] restoredData = new byte[originalLength];
            Array.Copy(data, restoredData, originalLength);
            return restoredData;
        }
        private void ProtectPasswordValuesInOperationalMemory()
        {
            encryptedPassword = AlignToBlockSize(encryptedPassword);
            password_salt = AlignToBlockSize(password_salt);
            password_IV = AlignToBlockSize(password_IV);
            password_masterKey =
AlignToBlockSize(password_masterKey);
            ProtectedMemory.Protect(encryptedPassword,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Protect(password_salt,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Protect(password_IV,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Protect(password_masterKey,
MemoryProtectionScope.SameLogon);
```

```
        }
        private void UnprotectPasswordValuesInOperationalMemory()
        {
            ProtectedMemory.Unprotect(encryptedPassword,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Unprotect(password_salt,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Unprotect(password_IV,
MemoryProtectionScope.SameLogon);
            ProtectedMemory.Unprotect(password_masterKey,
MemoryProtectionScope.SameLogon);
            encryptedPassword =
RestoreOriginalLength(encryptedPassword, encryptedPassword_len);
            password_salt = RestoreOriginalLength(password_salt,
password_salt_len);
            password_IV = RestoreOriginalLength(password_IV,
password_IV_len);
            password_masterKey =
RestoreOriginalLength(password_masterKey, password_masterKey_len);
        }
        private void setPassword(string password, MasterKey key)
        {
            if (password == null || password.Length == 0 || key ==
null || key._keyLen == 0)
            {
                throw new ArgumentNullException();
            }
            using (var aes = new AesManaged())
            {
                byte[] salt = new byte[8];

                using (var rng = new RNGCryptoServiceProvider())
                {
                    rng.GetBytes(salt);
//Здесь генерируется соль для пароля
                }

                aes.KeySize = 256;
                aes.BlockSize = 128;
                aes.Mode = CipherMode.CBC;
                aes.Padding = PaddingMode.ISO10126;
                key.unprotectKey();
                var keyGenerator = new
Rfc2898DeriveBytes(key.getKey(), salt, 1000);
                key.protectKey();
                aes.Key = keyGenerator.GetBytes(aes.KeySize / 8);
//Здесь генерируется Key для пароля
                aes.IV = keyGenerator.GetBytes(aes.BlockSize / 8);
//Здесь генерируется IV для пароля

                // Шифрование пароля
                using (var encryptor = aes.CreateEncryptor())
```

```
                {
                    byte[] passwordBytes =
System.Text.Encoding.UTF8.GetBytes(password);
                    encryptedPassword =
encryptor.TransformFinalBlock(passwordBytes, 0,
passwordBytes.Length);  //тут шифруется сам пароль с помощью
сгенерированных значений
                    encryptedPassword_len =
encryptedPassword.Length;
                    password_salt = salt;
                    password_salt_len = password_salt.Length;
                    password_IV = aes.IV;
                    password_IV_len = password_IV.Length;
                    password_masterKey = aes.Key;
                    password_masterKey_len =
password_masterKey.Length;
                    ProtectPasswordValuesInOperationalMemory();
                    Array.Clear(aes.Key, 0, aes.Key.Length);
                    Array.Clear(aes.IV, 0, aes.IV.Length);
                    Array.Clear(salt, 0, salt.Length);
                }
            }
        }
        private string getPassword()
        {
            if (encryptedPassword == null ||
encryptedPassword.Length == 0)//проверка, если что-то из этого
нулёвое, то расшифровки не получится
                throw new
ArgumentNullException(nameof(encryptedPassword));
            if (password_masterKey == null ||
password_masterKey.Length == 0)
                throw new
ArgumentNullException(nameof(password_masterKey));
            if (password_salt == null || password_salt.Length == 0)
                throw new
ArgumentNullException(nameof(password_salt));
            if (password_IV == null || password_IV.Length == 0)
                throw new
ArgumentNullException(nameof(password_IV));

            using (var aes = new AesManaged())
            {
                aes.KeySize = password_masterKey.Length * 8;
                aes.BlockSize = 128;
                aes.Mode = CipherMode.CBC;
                aes.Padding = PaddingMode.ISO10126;

                aes.Key = password_masterKey;
                aes.IV = password_IV;

                using (var decryptor = aes.CreateDecryptor())
```

```
                {
                    byte[] decryptedBytes =
decryptor.TransformFinalBlock(encryptedPassword, 0,
encryptedPassword.Length);
                    aes.Clear();
                    return Encoding.UTF8.GetString(decryptedBytes);
                }
            }
        }
        // Копируем пароль в буфер обмена и запускаем таймер
        private Thread clipboardThread;
        public void CopyPasswordToClipboard(int interval)
        {
            clipboardThread = new Thread(() =>
            {
                try
                {
                    Clipboard.Clear();
                    Timer clearClipboardTimer = new Timer();
                    clearClipboardTimer.Interval = interval;
                    UnprotectPasswordValuesInOperationalMemory();
                    Clipboard.SetText(getPassword());
                    ProtectPasswordValuesInOperationalMemory();
                    Thread.Sleep(interval);
                    Clipboard.Clear();
                    clipboardThread.Abort();
                }
                catch
                {
                    clipboardThread.Abort();
                }

            });
            clipboardThread.SetApartmentState(ApartmentState.STA);
            clipboardThread.Start();

        }
        public string getJsonPassword()
        {
            UnprotectPasswordValuesInOperationalMemory();
            string jPass = JsonConvert.SerializeObject(this);
            ProtectPasswordValuesInOperationalMemory();
            return jPass;
        }
        public void Dispose()
        {
            this.encryptedPassword = null;
            this.password_masterKey = null;
            if(clipboardThread!=null) clipboardThread.Abort();
        }
    }
}
```

Файл «*MasterKey.cs*»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace PasswordManager.Classes
{
    public class MasterKey
    {
        private byte[] _key;
        public int _keyLen;
        private const int blockSize = 16;
        public MasterKey(string key)
        {
            if (key.Length == 0)
            {
                this._keyLen = 0;
                this._key = null;
                return;
            }
            // Проверяем длину ключа
            if (key.Length % blockSize != 0)
            {
                int paddingSize = blockSize - (key.Length %
blockSize);
                _key = new byte[key.Length + paddingSize];

                // Копируем оригинальный ключ в выровненный массив
                Array.Copy(Encoding.UTF8.GetBytes(key), _key,
key.Length);
            }
            else
            {
                // Длина ключа уже кратна 16 байтам, не требуется
выравнивание
                _key = Encoding.UTF8.GetBytes(key);
            }
            _keyLen = key.Length;
            ProtectedMemory.Protect(_key,
MemoryProtectionScope.SameLogon);
        }
        public void unprotectKey()
        {
            ProtectedMemory.Unprotect(_key,
MemoryProtectionScope.SameLogon);
        }
        public void protectKey()
        {
```

```
            ProtectedMemory.Protect(_key,
MemoryProtectionScope.SameLogon);
        }
        public byte[] getKey()
        {
            return _key;
        }
    }
}
```

Файл «*JsonParser.cs*»

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PasswordManager.Classes
{
    public class JsonGroupEntryParser : IDisposable
    {
        public JObject GroupEntryToJObject(GroupEntry groupEntry)
        {
            if (groupEntry == null)
            {
                return null;
            }
            string json = JsonConvert.SerializeObject(groupEntry);
            var jObject = JObject.Parse(json);
            return jObject;

        }
        public GroupEntry JObjectToGroupEntry(JObject jObject)
        {
            if (jObject == null)
            {
                return null;
            }
            var groupEntry = new GroupEntry(jObject);
            return groupEntry;
        }
        public void Dispose()
        {
            return;
        }

    }
}
```

Файл «*GroupEntry.cs*»

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace PasswordManager
{
    public class GroupEntry : IDisposable
    {
        public List<GroupEntry> groupList;
        public List<AccountEntry> accountEntries;
        public string name;
        private GroupEntry()
        { }
        public GroupEntry(string name)
        {
            this.name = name;
            this.groupList = new List<GroupEntry>();
            this.accountEntries = new List<AccountEntry>();
        }

        public GroupEntry(JObject json)
        {
            try
            {
                GroupEntry groupEntry = new GroupEntry();
                groupEntry =
JsonConvert.DeserializeObject<GroupEntry>(json.ToString());
                this.groupList = groupEntry.groupList;
                this.accountEntries = groupEntry.accountEntries;
                this.name = groupEntry.name;
            }
            catch
            {
                MessageBox.Show("Ошибка при десереализации JSON-а
Возможно, файл поврежден", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }

        public void addAccountEntry(AccountEntry accountEntry)
        {
            accountEntries.Add(accountEntry);
        }
        public void removeAccountEntry(AccountEntry accountEntry)
        {
            accountEntry.Dispose();
            accountEntries.Remove(accountEntry);
        }
```

```
        public void modifyAccountEntry(AccountEntry oldAccountEntry,
AccountEntry newAccountEntry)
        {
            int index = accountEntries.FindIndex(entry => entry ==
oldAccountEntry);
            if (index != -1)
            {
                accountEntries[index] = newAccountEntry;
            }
        }
        public List<AccountEntry> GetAccountEntries()
        {
            if (accountEntries.Count != 0)
                return accountEntries;
            else return null;
        }

        public string getName()
        {
            return name;
        }

        public void Dispose()
        {
            for (int i = accountEntries.Count - 1; i >= 0; i--)
            {
                var obj = accountEntries[i];
                obj.Dispose();
                accountEntries.Remove(obj);
            }
            for (int i = groupList.Count - 1; i >= 0; i--)
            {
                var obj = groupList[i];
                obj.Dispose();
                groupList.Remove(obj);
            }
        }
    }
}
```

Файл «*Config.cs*»

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PasswordManager.Classes
{
    public class Config
    {
```

```
        public string databaseDirectory { get; set; }
        public string lastDbName { get; set; }
        public string pathWithName
        {
            get
            {
                return databaseDirectory + "\\" + lastDbName;
            }
            set
            {
                lastDbName = Path.GetFileName(value);
                databaseDirectory = Path.GetDirectoryName(value);
            }
        }
    }
}
```

Файл «*AccountEntry.cs*»

```
using Newtonsoft.Json;
using PasswordManager.Classes;
using System;
using System.Collections.Generic;

namespace PasswordManager
{
    ///<Summary>
    ///Класс представляет собой запись в листе паролей, содержащая в
себе сам пароль, логин и прочие поля (расширяется динамически)
    ///</Summary>
    public class AccountEntry : IDisposable
    {
        ///<Summary>
        ///Конструктор класса, который принимает на вход пароль,
логин (дополнительные поля добавляются по обращению к индексу)
        ///</Summary>
        public AccountEntry(string accountName, string login, string
password, MasterKey key, Dictionary<string, string> properties)
        {
            this.accountName = accountName;
            this.login = login;
            this.password = new Password(password, key);
            password = null;
            this.properties = properties;
        }

        ///<Summary>
        ///Конструктор класса, который принимает на вход пароль,
логин (дополнительные поля добавляются по обращению к индексу)
        ///</Summary>
        public AccountEntry(string accountName, string login,
Password password, Dictionary<string, string> properties)
        {
```

```
            this.accountName = accountName;
            this.login = login;
            this.password = password;
            this.properties = properties;
        }

        ///<Summary>
        ///Конструктор класса для Json
        ///</Summary>
        [JsonConstructor]
        private AccountEntry(string accountName, string login,
string password)
        {
            this.accountName = accountName;
            this.login = login;
            this.password =
JsonConvert.DeserializeObject<Password>(password); ;
            this.properties = new Dictionary<string, string>();
        }
        ///<Summary>
        ///Название аккаунта
        ///</Summary>
        public string accountName;

        ///<Summary>
        ///Логин приложения (мб почта или еще что нибудь)
        ///</Summary>
        public string login;

        ///<Summary>
        ///Сам пароль, который внутри уже сам шифруется у себя
        ///</Summary>
        [JsonIgnore]
        public Password password;

        ///<Summary>
        ///Необходимо для верного парсинга
        ///</Summary>
        [JsonProperty("Password")]
        private string SerializedPassword
        {
            get { return password.getJsonPassword(); }
        }
        ///<Summary>
        ///Словарь хранит в себе динамически расширяемые значения
полей
        ///</Summary>
        [JsonProperty]
        public Dictionary<string, string> properties { get; set; }

        ///<Summary>
```

```
        ///Обращение к полям записи (не получится обратиться к
паролю
        ///</Summary>
        public string this[string key]
        {
            get
            {
                string value;
                properties.TryGetValue(key, out value);
                return value;
            }
            set
            {
                properties[key] = value;
            }
        }

        public void Dispose()
        {
            password.Dispose();
        }
    }
}
```

Файл «*FileEncryptor.cs*»

```
using Newtonsoft.Json.Linq;
using PasswordManager.Classes;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace PasswordManager.Classes
{
    public class FileEncryptor
    {
        public static void EncryptAndSaveToFile(JObject jsonObj,
MasterKey key, string filePath)
        {
            // Сериализуем JObject в строку JSON
            string jsonStr = jsonObj.ToString();

            // Создаем новый экземпляр класса AES, используя мастер-
ключ в качестве ключа
            using (Aes aes = Aes.Create())
            {
                key.unprotectKey();
                aes.Key = key.getKey();
                key.protectKey();
```

```
                aes.IV = new byte[16]; // Начальное значение IV
должно быть случайным

                // Создаем объект Encryptor для AES, используя ключ
и IV
                ICryptoTransform encryptor =
aes.CreateEncryptor(aes.Key, aes.IV);

                // Создаем поток для записи зашифрованных данных в
файл
                using (FileStream fsEncrypt = new
FileStream(filePath, FileMode.Create))
                {
                    // Записываем начальное значение IV в файл
                    fsEncrypt.Write(aes.IV, 0, aes.IV.Length);

                    // Создаем поток для шифрования данных и записи
в выходной поток
                    using (CryptoStream csEncrypt = new
CryptoStream(fsEncrypt, encryptor, CryptoStreamMode.Write))
                    {
                        // Преобразуем строку JSON в байтовый массив
и записываем его в выходной поток
                        byte[] jsonBytes =
Encoding.UTF8.GetBytes(jsonStr);
                        csEncrypt.Write(jsonBytes, 0,
jsonBytes.Length);
                    }
                }
            }
        }
        public static JObject DecryptJsonFile(string filePath,
MasterKey key)
        {
            byte[] encryptedData = File.ReadAllBytes(filePath);
            byte[] iv = new byte[16];
            byte[] encryptedContent = new byte[encryptedData.Length
- iv.Length];

            Buffer.BlockCopy(encryptedData, 0, iv, 0, iv.Length);
            Buffer.BlockCopy(encryptedData, iv.Length,
encryptedContent, 0, encryptedContent.Length);

            using (var aes = Aes.Create())
            {
                key.unprotectKey();
                aes.Key = key.getKey();
                key.protectKey();
                aes.IV = iv;

                using (var decryptor = aes.CreateDecryptor())
```

```
                using (var memoryStream = new
MemoryStream(encryptedContent))
                using (var cryptoStream = new
CryptoStream(memoryStream, decryptor, CryptoStreamMode.Read))
                using (var streamReader = new
StreamReader(cryptoStream))
                {
                    string decryptedJson = streamReader.ReadToEnd();
                    JObject jsonObject =
JObject.Parse(decryptedJson);
                    return jsonObject;
                }
            }
        }
    }
}
```

Файл «*FileManager.cs*»

```
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net.NetworkInformation;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml.Linq;

namespace PasswordManager.Classes
{
    public class FileManager
    {
        private const string configFilePath = "passwdConfig.cfg";
        public static GroupEntry LoadDb(string filePath, MasterKey
key)
        {
            var jObject = FileEncryptor.DecryptJsonFile(filePath,
key);

            var groupEntry = new
JsonGroupEntryParser().JObjectToGroupEntry(jObject);
            return groupEntry;
        }
        public static void SaveDbToFile(GroupEntry groupEntry,
string filePath, MasterKey key)
        {
            var json = new
JsonGroupEntryParser().GroupEntryToJObject(groupEntry);
            FileEncryptor.EncryptAndSaveToFile(json, key, filePath);
        }
```

```
        public static Config LoadOrCreateConfig()
        {
            if (File.Exists(configFilePath))
            {
                // Файл конфигурации уже существует, считываем его
содержимое
                string configFileContent =
File.ReadAllText(configFilePath);
                return
JsonConvert.DeserializeObject<Config>(configFileContent);
            }
            else
            {
                string selectedPath = null;
                // Создание экземпляра диалогового окна выбора пути
до директории
                using (var folderDialog = new FolderBrowserDialog())
                {
                    // Установка заголовка окна
                    folderDialog.Description = "Выберите директорию
для сохранения баз данных";

                    // Отображение диалогового окна и проверка
результата
                    if (folderDialog.ShowDialog() ==
DialogResult.OK)
                    {
                        // Получение выбранного пути до директории
                        selectedPath = folderDialog.SelectedPath;

                    }
                }
                Config config = new Config();
                config.databaseDirectory = selectedPath;
                string configJson =
JsonConvert.SerializeObject(config);
                File.WriteAllText(configFilePath, configJson);
                return config;
            }
        }
        public static void SaveConfigFile(Config config)
        {
            string configJson = JsonConvert.SerializeObject(config);
            File.WriteAllText(configFilePath, configJson);
        }
    }
}
```

Файл «*MainForm.cs*»

```
using Newtonsoft.Json.Linq;
using PasswordManager.Classes;
using PasswordManager.Forms;
```

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Text;
using System.Globalization;
using System.IO;
using System.Runtime.InteropServices;
using System.Runtime.InteropServices.ComTypes;
using System.Security.Cryptography;
using System.Security.Principal;
using System.Windows.Forms;
using System.Xml.Linq;

namespace PasswordManager
{
    public partial class MainForm : Form
    {
        MasterKey key;
        Config config;
        bool isNewDb = false;
        public MainForm()
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            MainForm_Init();
            config = FileManager.LoadOrCreateConfig();
            if (!ShowLoginForm(config.pathWithName))
            {
                this.Close();
                return;
            }
            if (isNewDb)
            {
                this.Text = "Passwd - " + config.lastDbName;
                return;
            }
            try
            {
                var newRoot =
FileManager.LoadDb(config.pathWithName, key);
                rootEntry = newRoot;
                treeViewManager.Override(rootEntry);
                listViewManager.setNewGroup(rootEntry);
            }
            catch (Exception ex)
            {
                var dialogResult = MessageBox.Show("Ошибка при
открытии базы паролей\nЖелаете создать новую базу паролей?",
                    "Ошибка",
                    MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning);
                if (dialogResult == DialogResult.OK)
```

```
                {
                    if (!ShowNewDbForm())
                    {
                        this.Close();
                    }
                }
                else this.Close();
            }
            this.Text = "Passwd - " + config.lastDbName;
        }
        private bool ShowLoginForm(string fileName = null, bool
showNewDbButton = true)
        {
            using (LoginForm loginForm = new LoginForm(fileName,
showNewDbButton))
            {
                var dialogResult = loginForm.ShowDialog();
                if (dialogResult == DialogResult.OK)
                {
                    this.key = new MasterKey(loginForm.key);
                    config.pathWithName = loginForm.FileName;
                }
                else if (dialogResult == DialogResult.Yes)  //из-за
неимением большего используем Yes, а так то результат NewDb
                {
                    if (!ShowNewDbForm())
                    {
                        return false;
                    }
                }
                else
                {
                    return false;
                }
                return true;
            }
        }
        private bool ShowNewDbForm()
        {
            using (NewDbForm newDbForm = new NewDbForm())
            {
                var dialogResult = newDbForm.ShowDialog();
                if (dialogResult != DialogResult.OK)
                {
                    return false;
                }
                this.key = newDbForm.key;
                config.lastDbName = newDbForm.dbName;
                isNewDb = true;
                return true;
            }
        }
```

```
        private void MainForm_Init()
        {
            tbSearch_init();
            lwAccounts_init();
            twGroups_init();
        }

        #region lwAccounts
        private void lwAccounts_init()
        {
            listViewManager = new ListViewManager(lwAccounts,
tslblAccountsCount);
        }

        ListViewManager listViewManager;
        public class ListViewManager
        {
            private ListView _lwAccounts;
            private GroupEntry _groupEntry;
            private ToolStripLabel _lblAccountsCount;
            public ListViewManager(ListView listView, ToolStripLabel
lblAccountsCount)
            {
                this._lwAccounts = listView;
                _lwAccounts.Columns.Add("Название", 150);
                _lwAccounts.Columns.Add("Логин", 150);
                _lwAccounts.Columns.Add("Пароль", 50);
                _lwAccounts.View = View.Details;
                _lwAccounts.Scrollable = true;
                _lwAccounts.HeaderStyle =
ColumnHeaderStyle.Clickable;
                _lblAccountsCount = lblAccountsCount;
            }
            public void AddAccountEntry(AccountEntry account)
            {
                _groupEntry.addAccountEntry(account);
                PopulateListView();
            }
            public void RemoveSelectedAccountEntry()
            {
                if (_lwAccounts.SelectedItems.Count != 0)
                {

                    var accountEntry =
_lwAccounts.SelectedItems[0].Tag as AccountEntry;
                    _groupEntry.removeAccountEntry(accountEntry);
                }
                PopulateListView();
            }
            public void setNewGroup(GroupEntry group)
            {
                this._groupEntry = group;
```

```
                PopulateListView();
            }
            public void PopulateListView()
            {
                List<AccountEntry> accountEntries =
_groupEntry.GetAccountEntries();
                // Очистка ListView перед добавлением новых данных
                _lwAccounts.Items.Clear();

                if (accountEntries == null)
                {
                    // Добавление сообщения о пустом списке в
ListView
                    ListViewItem emptyItem = new
ListViewItem("Записей не найдено");
                    _lwAccounts.Items.Add(emptyItem);
                    _lblAccountsCount.Text = "0 аккаунт(ов)";
                    return;
                }
                // Проход по списку AccountEntry и добавление данных
в ListView
                foreach (AccountEntry entry in accountEntries)
                {
                    // Создание новой строки для ListView
                    ListViewItem item = new
ListViewItem(entry.accountName);

                    // Добавление логина во вторую колонку
                    item.SubItems.Add(entry.login);

                    // Добавление пароля в третью колонку
                    item.SubItems.Add("********");

                    // Установка Tag в качестве ссылки на
соответствующий объект AccountEntry
                    item.Tag = entry;

                    // Добавление строки в ListView
                    _lwAccounts.Items.Add(item);
                }
                _lblAccountsCount.Text =
_lwAccounts.Items.Count.ToString() + " аккаунт(ов)";
            }
            public void ModifySelectedAccountEntry(AccountEntry
newEntry)
            {
                if (_lwAccounts.SelectedItems.Count != 0)
                {
                    var oldEntry = _lwAccounts.SelectedItems[0].Tag
as AccountEntry;
                    _groupEntry.modifyAccountEntry(oldEntry,
newEntry);
```

```
                }
                PopulateListView();
            }
        }

        private void btnDeleteAccount_Click(object sender, EventArgs
e)
        {
            listViewManager.RemoveSelectedAccountEntry();
        }
        private void btnAdd_Click(object sender, EventArgs e)
        {
            using (AddAccountForm addAccountForm = new
AddAccountForm(key))
            {
                if (addAccountForm.ShowDialog() == DialogResult.OK)
                {

listViewManager.AddAccountEntry(addAccountForm.account);
                }
            }
        }
        #endregion

        #region tbSearch

        const string DEFUALT_SEARCH_TEXT = "Поиск...";
        private void tbSearch_init()
        {
            tbSearch.Text = DEFUALT_SEARCH_TEXT;
            tbSearch.ForeColor = SystemColors.GrayText;
        }

        private void tbSearch_GotFocus(object sender, EventArgs e)
        {
            // Очистить TextBox при получении фокуса
            if (tbSearch.Text == DEFUALT_SEARCH_TEXT)
            {
                tbSearch.Text = "";
                tbSearch.ForeColor = SystemColors.WindowText;
            }
        }
        private void tbSearch_LostFocus(object sender, EventArgs e)
        {
            // Восстановить текст по умолчанию, если TextBox пуст
            if (string.IsNullOrWhiteSpace(tbSearch.Text))
            {
                tbSearch.Text = DEFUALT_SEARCH_TEXT;
                tbSearch.ForeColor = SystemColors.GrayText;
                listViewManager.PopulateListView();
            }
        }
```

```
        private void tbSearch_TextChanged(object sender, EventArgs
e)
        {
            if (tbSearch.Text == "")
            {
                return;
            }
            string searchText = tbSearch.Text;
            lwAccounts.Items.Clear(); // Очистка ListView перед
добавлением результатов поиска
            SearchNodes(twGroups.Nodes, searchText);
        }

        private void SearchNodes(TreeNodeCollection nodes, string
searchText)
        {
            foreach (TreeNode node in nodes)
            {
                var groupEntry = node.Tag as GroupEntry;
                if (groupEntry.accountEntries != null &&
groupEntry.accountEntries.Count != 0)
                {
                    foreach (AccountEntry entry in
groupEntry.accountEntries)
                    {
                        if (entry.accountName.Contains(searchText))
                        {
                            // Создание ListViewItem для отображения
AccountEntry в ListView
                            ListViewItem item = new
ListViewItem(entry.accountName);
                            item.SubItems.Add(entry.login);
                            item.SubItems.Add("********");
                            item.Tag = entry;
                            // Добавление ListViewItem в ListView
                            lwAccounts.Items.Add(item);
                        }
                    }
                }
                // Рекурсивный вызов функции для поиска в дочерних
нодах
                if (node.Nodes.Count != 0) SearchNodes(node.Nodes,
searchText);
            }
        }

        #endregion

        #region twGroups
        GroupEntry rootEntry;
        TreeViewManager treeViewManager;
```

```
        public class TreeViewManager
        {
            private TreeView _treeView;
            public TreeViewManager(TreeView treeView)
            {
                _treeView = treeView;
            }

            public void AddGroupEntry(GroupEntry groupEntry,
TreeNode parentNode = null)
            {
                TreeNode node = new TreeNode(groupEntry.name);
                node.Tag = groupEntry;

                // Добавляем узел в дерево
                if (parentNode == null)
                    _treeView.Nodes.Add(node);
                else
                    parentNode.Nodes.Add(node);

                // Рекурсивно добавляем поддеревья
                if (groupEntry.groupList != null)
                {
                    foreach (var childEntry in groupEntry.groupList)
                    {
                        AddGroupEntry(childEntry, node);
                    }
                }
            }
            public void InsertTreeNode(GroupEntry newGroupEntry,
TreeNode parentNode = null)
            {

                // Создаем новый узел и устанавливаем его свойство
Tag
                TreeNode newNode = new TreeNode(newGroupEntry.name);
                newNode.Tag = newGroupEntry;

                if (parentNode == null)
                {
                    // Добавляем новый узел в коллекцию узлов
родительского узла
                    _treeView.Nodes.Add(newNode);
                }
                else
                {
                    // Добавляем новый GroupEntry в коллекцию
Children родительского GroupEntry
                    var parentEntry = parentNode.Tag as GroupEntry;
                    parentEntry.groupList.Add(newGroupEntry);
                    parentNode.Nodes.Add(newNode);
```

```
            }
            _treeView.ExpandAll();
        }
        public void RemoveSelectedGroupEntry()
        {
            TreeNode node = _treeView.SelectedNode;
            TreeNode rootNode = _treeView.Nodes[0];
            if (node == null)
            {
                return;
            }
            if (rootNode == node)
            {
                return;
            }
            var entry = node.Tag as GroupEntry;
            entry.Dispose();
            node.Remove();
        }
        public void Override(GroupEntry groupEntry)
        {
            _treeView.Nodes.Clear();
            AddGroupEntry(groupEntry, null);
            _treeView.ExpandAll();
        }
    }
    private void twGroups_AfterSelect(object sender,
TreeViewEventArgs e)
    {
        if (e.Node == null || e.Node.Tag == null)
        {
            return;
        }
        TreeNode selectedNode = twGroups.SelectedNode;
        if (selectedNode != null)
        {
            // Получение связанного с узлом объекта GroupEntry
            GroupEntry selectedGroup = selectedNode.Tag as
GroupEntry;

            // Проверка, что связанный объект GroupEntry не
является null
            if (selectedGroup != null)
            {
                // Вызов метода PopulateListView с
соответствующим списком AccountEntry
                listViewManager.setNewGroup(selectedGroup);
            }
        }
    }
    private void twGroups_init()
    {
```

```
            rootEntry = new GroupEntry("Группы");
            treeViewManager = new TreeViewManager(twGroups);
            treeViewManager.InsertTreeNode(rootEntry);
        }
        private void btnPlus_Click(object sender, EventArgs e)
        {
            using (AddGroupForm inputDialog = new AddGroupForm())
            {
                if (inputDialog.ShowDialog() == DialogResult.OK)
                {
                    // Получаем выбранный узел и добавляем туда
новую запись, с именем из диалогового окна
                    TreeNode selectedNode = twGroups.SelectedNode;
                    treeViewManager.InsertTreeNode(new
GroupEntry(inputDialog.groupName),
                        (selectedNode != null) ? selectedNode :
null);
                }
            }
        }
        private void btnMinus_Click(object sender, EventArgs e)
        {
            treeViewManager.RemoveSelectedGroupEntry();
        }
        #endregion
        private void открытьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (!ShowLoginForm(null, false))
            {
                return;
            }
            try
            {
                var newRoot =
FileManager.LoadDb(config.pathWithName, key);
                rootEntry = newRoot;
                treeViewManager.Override(rootEntry);
                listViewManager.setNewGroup(rootEntry);
            }
            catch
            {
                MessageBox.Show("Неправильный мастер-ключ.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
        }
        private void lwAccounts_MouseDoubleClick(object sender,
MouseEventArgs e)
        {
            if (lwAccounts.SelectedItems.Count > 0)
            {
```

```
                ListViewItem selectedItem =
lwAccounts.SelectedItems[0];
                // Получаем объект из свойства Tag выбранного
элемента
                var selectedObject = selectedItem.Tag as
AccountEntry;
                if (selectedObject == null)
                {
                    toolStripStatusLabel.Text = "Невозможно
скопировать пароль. Нет записей";
                    return;
                }

selectedObject.password.CopyPasswordToClipboard(3000);
                toolStripStatusLabel.Text = "Пароль от аккаунта " +
selectedObject.accountName + " скопирован в буфер обмена на 3
секунды";
            }
        }
        private void lwAccounts_MouseClick(object sender,
MouseEventArgs e)
        {
            if (e.Button == MouseButtons.Right)
            {
                var accountEntry = lwAccounts?.SelectedItems[0].Tag
as AccountEntry;
                if (accountEntry == null)
                {
                    return;
                }
                using (AddAccountForm addAccountForm = new
AddAccountForm(accountEntry, key))
                {
                    if (addAccountForm.ShowDialog() ==
DialogResult.OK)
                    {

listViewManager.ModifySelectedAccountEntry(addAccountForm.account);
                    }
                }
            }
        }

        //обработка горячих клавиш
        private void MainForm_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.Control && e.KeyCode == Keys.S)       // Ctrl-S
Save
            {
                FileManager.SaveDbToFile(rootEntry,
config.pathWithName, key);
                toolStripStatusLabel.Text = "Сохранено!";
```

```
            }
        }
        private void btnSave_Click(object sender, EventArgs e)
        {
            FileManager.SaveDbToFile(rootEntry, config.pathWithName,
key);
            toolStripStatusLabel.Text = "Сохранено!";
        }
        private void MainForm_FormClosing(object sender,
FormClosingEventArgs e)
        {

            DialogResult dialogResult = MessageBox.Show("Сохранить
текущую базу паролей?", "Выход", MessageBoxButtons.YesNo);
            if (dialogResult == DialogResult.Yes)
            {
                FileManager.SaveConfigFile(config);
                FileManager.SaveDbToFile(rootEntry,
config.pathWithName, key);
            }

        }
        private void СохранитьToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            FileManager.SaveDbToFile(rootEntry, config.pathWithName,
key);
            toolStripStatusLabel.Text = "Сохранено!";
        }
        private void новыйФайлToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (!ShowNewDbForm())
            {
                return;
            }
            rootEntry = new GroupEntry("Группы");
            treeViewManager.Override(rootEntry);
            toolStripStatusLabel.Text = "Создана новая база
паролей";
        }
        private void оПрограммеToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            using (AboutProgramForm form = new AboutProgramForm())
            {
                form.ShowDialog();
            }
        }
    }
}
```

Файл «*AboutProgramForm.cs*»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Drawing;
using System.Linq;
using System.Reflection;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    partial class AboutProgramForm : Form
    {
        public AboutProgramForm()
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            this.Text = String.Format("About {0}", AssemblyTitle);
            this.labelProductName.Text = "Passwd - Менеджер
паролей";
            this.labelVersion.Text = String.Format("Version {0}",
AssemblyVersion);
            this.labelCopyright.Text = "Создатель: Лобанов Дмитрий";
            this.labelCompanyName.Text = "ПГУ, факультет
вычислительной техники";
            this.textBoxDescription.Text = "Менеджер паролей
предназначен для безопасного хранения паролей локально и управления
базой паролей";
        }

        #region Assembly Attribute Accessors

        public string AssemblyTitle
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyT
itleAttribute), false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute titleAttribute =
(AssemblyTitleAttribute)attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
```

```
                return
System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAsse
mbly().CodeBase);
            }
        }

        public string AssemblyVersion
        {
            get
            {
                return
Assembly.GetExecutingAssembly().GetName().Version.ToString();
            }
        }

        public string AssemblyDescription
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyD
escriptionAttribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return
((AssemblyDescriptionAttribute)attributes[0]).Description;
            }
        }

        public string AssemblyProduct
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyP
roductAttribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return
((AssemblyProductAttribute)attributes[0]).Product;
            }
        }

        public string AssemblyCopyright
        {
            get
            {
```

```
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyC
opyrightAttribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return
((AssemblyCopyrightAttribute)attributes[0]).Copyright;
            }
        }

        public string AssemblyCompany
        {
            get
            {
                object[] attributes =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(AssemblyC
ompanyAttribute), false);
                if (attributes.Length == 0)
                {
                    return "";
                }
                return
((AssemblyCompanyAttribute)attributes[0]).Company;
            }
        }
        #endregion
    }
}
```

Файл «*AddAccountForm.cs*»

```
using PasswordManager.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    public partial class AddAccountForm : Form
    {
        public AccountEntry account;
        MasterKey key;
        Dictionary<string, string> properties;

        public AddAccountForm(MasterKey key)
```

```
        {
            this.key = key;
            account = null;
            this.properties = new Dictionary<string, string>();
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            txtPassword.UseSystemPasswordChar = true;
            // Установка шрифта кнопки, поддерживающего символы
Unicode
            btnShowPassword.Font = new Font("Segoe UI Emoji", 8);
            // Установка текста кнопки на символ глаза
            btnShowPassword.Text = "\U0001F441";

        }
        public AddAccountForm(AccountEntry account, MasterKey key)
        {
            this.properties = new Dictionary<string, string>();
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            txtPassword.UseSystemPasswordChar = true;
            // Установка шрифта кнопки, поддерживающего символы
Unicode
            btnShowPassword.Font = new Font("Segoe UI Emoji", 8);
            // Установка текста кнопки на символ глаза
            btnShowPassword.Text = "\U0001F441";

            txtAccountName.Text = account.accountName;
            txtLogin.Text = account.login;
            txtPassword.Text = "Измените это поле, чтобы поменять
пароль";
            this.account = account;
            this.key = key;
            foreach (var property in account.properties)
            {
                // Создание новых элементов TextBox
                TextBox keyTextBox = new TextBox();
                TextBox valueTextBox = new TextBox();

                // Установка положения и размеров TextBox
                keyTextBox.Location = new Point(1,
pProperties.Controls.Count * 10); // Расположение по вертикали,
учитывая уже имеющиеся элементы
                valueTextBox.Location = new Point(101,
pProperties.Controls.Count * 10); // Расположение по вертикали,
учитывая уже имеющиеся элементы
                keyTextBox.Size = new Size(100, 20);
                valueTextBox.Size = new Size(190, 20);

                keyTextBox.Text = property.Key;
                valueTextBox.Text = property.Value;
                // Добавление TextBox в Panel
                pProperties.Controls.Add(keyTextBox);
```

```
                pProperties.Controls.Add(valueTextBox);
            }
        }

        private void btnAddProperty_Click(object sender, EventArgs
e)
        {
            // Создание новых элементов TextBox
            TextBox keyTextBox = new TextBox();
            TextBox valueTextBox = new TextBox();

            // Установка положения и размеров TextBox
            keyTextBox.Location = new Point(1,
pProperties.Controls.Count * 10); // Расположение по вертикали,
учитывая уже имеющиеся элементы
            valueTextBox.Location = new Point(101,
pProperties.Controls.Count * 10); // Расположение по вертикали,
учитывая уже имеющиеся элементы
            keyTextBox.Size = new Size(100, 20);
            valueTextBox.Size = new Size(190, 20);

            // Добавление TextBox в Panel
            pProperties.Controls.Add(keyTextBox);
            pProperties.Controls.Add(valueTextBox);
        }

        private void btnShowPassword_MouseDown(object sender,
MouseEventArgs e)
        {
            txtPassword.UseSystemPasswordChar = false;

        }

        private void btnShowPassword_MouseUp(object sender,
MouseEventArgs e)
        {
            txtPassword.UseSystemPasswordChar = true;

        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            if (txtPassword.TextLength == 0 ||
txtAccountName.TextLength == 0)
            {
                MessageBox.Show("Поле Пароль и Название аккаунта
должны быть заполнены", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }

            for (int i = 0; i < pProperties.Controls.Count; i += 2)
```

```
            {
                if (pProperties.Controls[i] is TextBox keyTextBox &&
pProperties.Controls[i + 1] is TextBox valueTextBox)
                {
                    string key = keyTextBox.Text;
                    string value = valueTextBox.Text;
                    properties.Add(key, value);
                }
            }

            if (account != null)
            {
                if (txtPassword.Text == "Измените это поле, чтобы
поменять пароль")
                {
                    account = new AccountEntry(txtAccountName.Text,
txtLogin.Text, account.password, properties);
                    DialogResult = DialogResult.OK;
                    return;
                }
            }
            account = new AccountEntry(txtAccountName.Text,
txtLogin.Text, txtPassword.Text, key, properties);
            //groupName = txtGroupName.Text;
            DialogResult = DialogResult.OK;
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }
    }
}
```

Файл «*AddGroupForm.cs*»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    public partial class AddGroupForm : Form
    {
        public string groupName;
        public AddGroupForm()
        {
```

```
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            // Закрываем форму
            DialogResult = DialogResult.Cancel;
        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            if (txtGroupName.Text.Length == 0)
            {
                MessageBox.Show("Некорректный ввод!",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            // Закрываем форму
            groupName = txtGroupName.Text;
            DialogResult = DialogResult.OK;
        }
    }
}
```

Файл «*LoginForm.cs*»

```
using PasswordManager.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    public partial class LoginForm : Form
    {
        public string key;
        private string _fileName;
        public string FileName {
            get
            {
                return _fileName;
            }
            set
            {
                _fileName = value;
                txtFileName.Text = value;
```

```
            }
        }
        public LoginForm(string fileName = null, bool
showNewDbButton = true)
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
            if (showNewDbButton == false)
            {
                btnNewDB.Enabled = false;
            }
            if (fileName != null)
            {
                this.FileName = fileName;
            }
        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            if (txtKey.TextLength == 0 || txtFileName.TextLength ==
0)
            {
                return;
            }
            this.key = txtKey.Text;
            DialogResult = DialogResult.OK;
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }

        private void btnNewDB_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Yes;
        }

        private void btnPath_Click(object sender, EventArgs e)
        {
            // Создание экземпляра диалогового окна выбора пути до
директории
            using (var fileDialog = new OpenFileDialog())
            {
                DialogResult result = fileDialog.ShowDialog();
                // Отображение диалогового окна и проверка
результата
                if (result == DialogResult.OK)
                {
                    // Получение выбранного пути до директории
                    this.FileName = fileDialog.FileName;
                }
```

```
            }
        }

        private void LoginForm_KeyDown(object sender, KeyEventArgs
e)
        {
            if (e.KeyCode == Keys.Enter)
            {
                btnOk_Click(sender, e);
            }
        }
    }
}
```

Файл «*NewDbForm.cs*»

```
using PasswordManager.Classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    public partial class NewDbForm : Form
    {
        public MasterKey key;
        public string dbName;
        public NewDbForm()
        {
            InitializeComponent();
            this.StartPosition = FormStartPosition.CenterScreen;
        }

        private void btnOk_Click(object sender, EventArgs e)
        {
            if (txtMasterKey.TextLength == 0 || txtName.TextLength
== 0)
            {
                MessageBox.Show("Необходимо заполнить поля!",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Information);
                return;
            }

            key = new MasterKey(txtMasterKey.Text);
            dbName = txtName.Text;
            DialogResult = DialogResult.OK;
```

```
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
        }
    }
}
```

Файл «*GeneratePasswordForm.cs*»

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PasswordManager.Forms
{
    public partial class GeneratePasswordForm : Form
    {

        public GeneratePasswordForm()
        {
            InitializeComponent();
        }
        public class PasswordGenerator
        {
            private const string LowercaseChars =
"abcdefghijklmnopqrstuvwxyz";
            private const string UppercaseChars =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            private const string NumericChars = "0123456789";
            private const string SpecialChars = "!@#$%^&*()-
=_+[]{}|;:,.<>?";
            public static string GeneratePassword(int length, bool
includeLowercase, bool includeUppercase, bool includeNumeric, bool
includeSpecial)
            {
                string charSet = "";

                if (includeLowercase)
                    charSet += LowercaseChars;

                if (includeUppercase)
                    charSet += UppercaseChars;
```

```
                if (includeNumeric)
                    charSet += NumericChars;

                if (includeSpecial)
                    charSet += SpecialChars;

                StringBuilder password = new StringBuilder();
                Random random = new Random();

                for (int i = 0; i < length; i++)
                {
                    int randomIndex = random.Next(0,
charSet.Length);
                    password.Append(charSet[randomIndex]);
                }

                return password.ToString();
            }
        }
        private void btnGenerate_Click(object sender, EventArgs e)
        {
            int generatedPasswordLen = 0;
            try
            {
                generatedPasswordLen = int.Parse(tbLength.Text);
            }
            catch
            {
                MessageBox.Show("Вы написали в поле \"Длина\" что-
то, кроме цифр", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                return;
            }
            if (generatedPasswordLen < 0 || generatedPasswordLen >
128)
            {
                MessageBox.Show("Не стоит генерировать длину, больше
128", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
            tbPassword.Text =
PasswordGenerator.GeneratePassword(generatedPasswordLen,
cbLowerCase.Checked, cbUpperCase.Checked, cbNumerals.Checked,
cbSymblos.Checked);
        }
    }
}
```

Приложение В

(Обязательное)

Менеджер паролей

Презентация

Тема работы:
«Менеджер паролей»

Автор работы: студент группы 19ВВП1 Лобанов Д.В.
Научный руководитель: к.т.н. доцент каф. ВТ ПГУ Митрохин М.А.

Пенза 2023

Цель работы

Цель работы

Разработать приложение, которое будет осуществлять безопасное хранение, редактирование и изменение паролей пользователя.

Объект работы

Приложение, которое выполняет следующие функции:

- Запись новых паролей в базу паролей
- Редактирование записей в базе
- Загрузка базы паролей из файла
- Сохранение базы паролей в файл
- Генерация новых паролей

Почему данная тема актуальна

Каждый пользователь хранит свои пароли по-разному.

Варианты хранения паролей:

- В голове
- В тетради/на листке/в блокноте
- В текстовом файле на компьютере

3

Аналоги

4

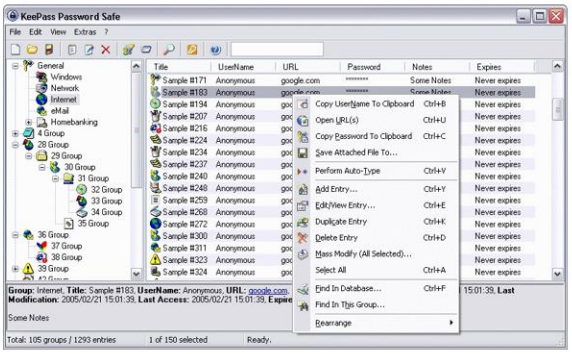
KeePass

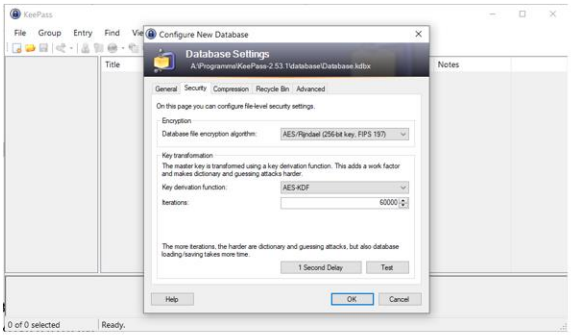
Плюсы:

- Бесплатный
- Открытый исходный код
- Хранение паролей на компьютере
- Поддержка дополнительных плагинов

Минусы:

- Сложность в освоении
- Чрезмерная перегрузка функционалом
- Долгая настройка





5

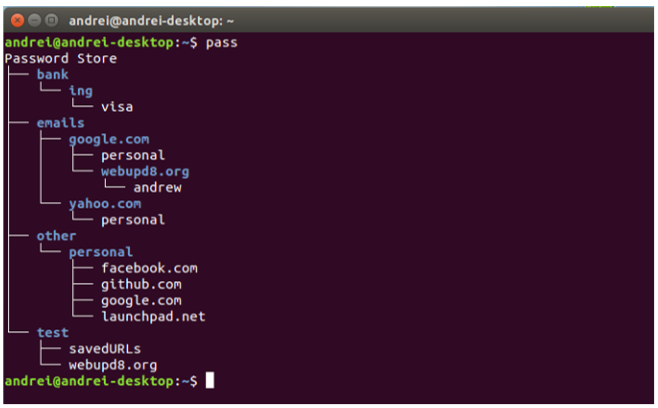
Pass

Плюсы:

- Бесплатный
- Открытый исходный код
- Хранение паролей на компьютере

Минусы:

- Отсутствие приложения на windows
- Отсутствие графического интерфейса



6

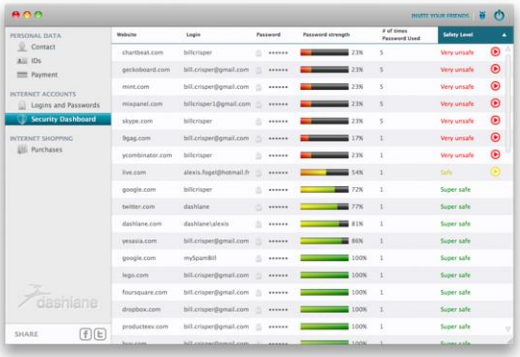
Dashlane

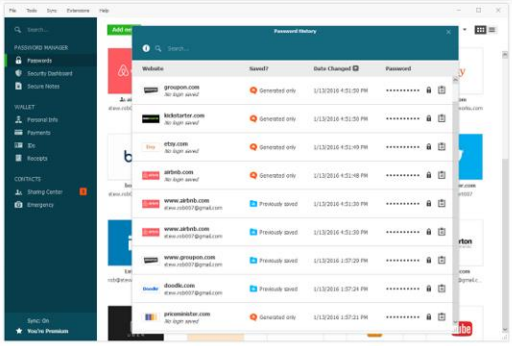
Плюсы:

- Интуитивно понятный интерфейс
- Автозаполнение форм паролей

Минусы:

- Платная модель подписки
- Зависимость от облачного хранения



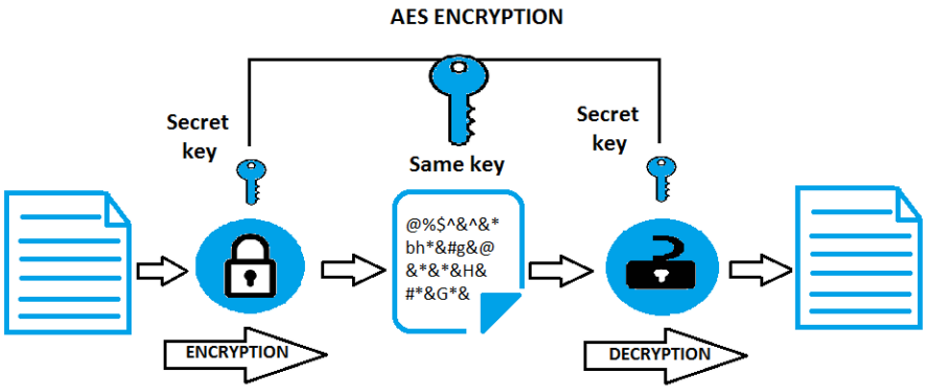


7

Стек технологий

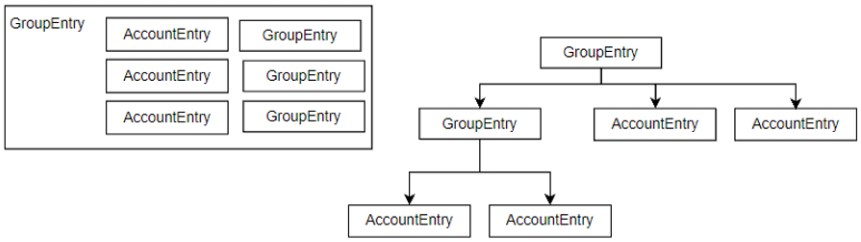
8

Безопасность при запущенном приложении



9

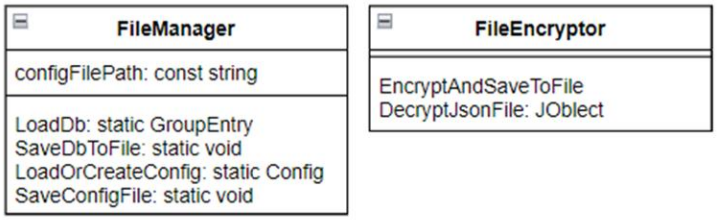
Разработка удобной архитектуры хранения информации



10

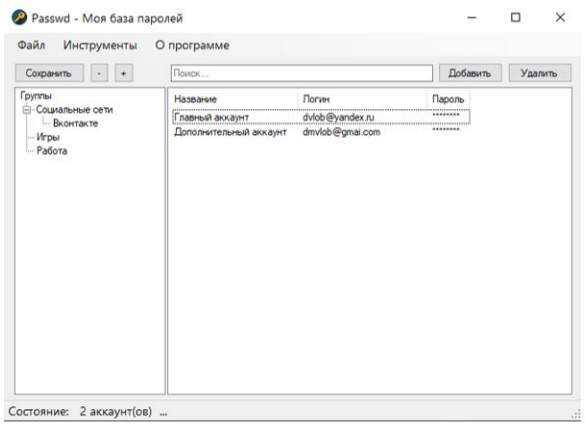
Работа с файлами

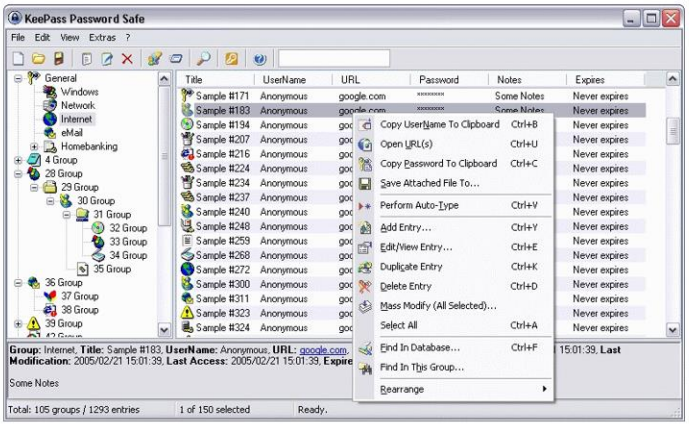
Для работы с файлами были реализованы 2 класса: FileManager и FileEncryptor.



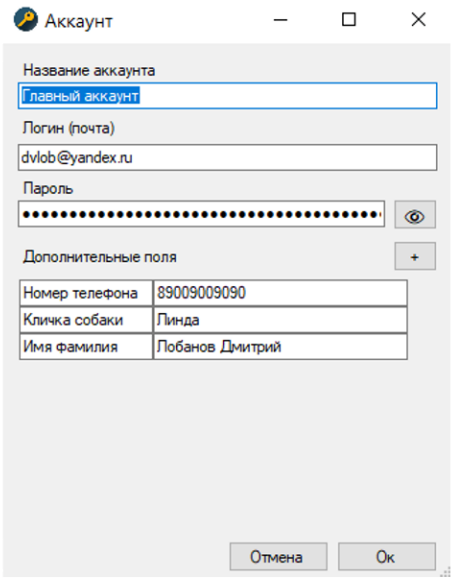
11

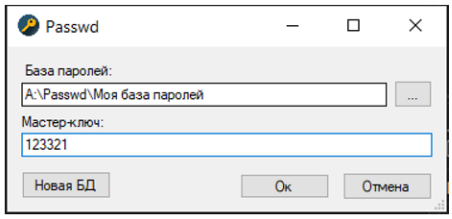
Графический интерфейс





12





13

Заключение

- В ходе выполнения выпускной квалификационной работы был разработан инструмент хранения, редактирования и генерирования паролей, с использованием языка C#, как основного языка программирования.
- Приложение предоставляет пользователю возможность импортировать и хранить информацию о пользовательских учетных записях, организовывать пароли по категориям и выполнять поиск и фильтрацию для удобного доступа.
- Данная работа открывает перспективы для импортирования разработанной технологии менеджера паролей на мобильные устройства, такие как смартфоны и планшеты.

14