**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет Кибербезопасности

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

**ОТЧЕТ ПО КУРСОВОЙ РАБОТЕ**

Разработка менеджера паролей

(тема работы)

Информационная безопасность 10.03.01

(код и наименование направления/специальности)

Студент:

Михайлов И.Д, ИКБ-32

(Ф.И.О., № группы) (подпись)

Преподаватель:

(уч. степень, уч. звание, Ф.И.О.) (подпись)

# Оглавление

[Оглавление 2](#_Toc5)

[Введение 2](#_Toc6)

[Основная часть 4](#_Toc7)

[1. Теоретическая часть 4](#_Toc8)

[2. Постановка задачи 5](#_Toc9)

[3. Архитектура и структура приложения 9](#_Toc10)

[3.1. Структура проекта 9](#_Toc11)

[3.2. Графический интерфейс 9](#_Toc12)

[3.3 Блок-схема приложения 10](#_Toc13)

[3.3.1 Блок-схема алгоритма 10](#_Toc14)

[3.3.1 Блок-схема методов 10](#_Toc15)

[3.4.1 Диаграмма классов 12](#_Toc16)

[3.4.2 Диаграмма прецедентов 12](#_Toc17)

[3.5 Положительные и негативные примеры работы приложения 13](#_Toc18)

[3.5.1 Положительные примеры 13](#_Toc19)

[3.5.2 Негативные примеры 13](#_Toc20)

[3.6 Библиотеки, сторонние классы и интерфейсы, которые используются в коде 13](#_Toc21)

[3.6.1. javax.swing.\* (Java Swing) 13](#_Toc22)

[3.6.2. org.apache.logging.log4j.\* (Apache Log4j) 14](#_Toc23)

[3.6.4. java.security.\* (Java Security API) 14](#_Toc24)

[3.6.5. java.util.\* (Java Utilities) 15](#_Toc25)

[3.6.6. java.awt.event.\* (Java AWT Event Handling) 15](#_Toc26)

[3.6.7. java.lang.\* (Java Core) 15](#_Toc27)

[3.6.8. javax.swing.event.\* (Swing Event Handling) 15](#_Toc28)

[4. Реализация шифрования 16](#_Toc29)

[4.1. Шифрование Base64 16](#_Toc30)

[4.2. Хэширование MD5 16](#_Toc31)

[4.3. Шифр Фейстеля 17](#_Toc32)

[4.4. Шифрование с солью (Salted Base64) 17](#_Toc33)

[5. Обработка ошибок и логирование при помощи log4j 17](#_Toc34)

[6. Сборка gradle 18](#_Toc35)

[Заключение 18](#_Toc36)

[Публикация приложения 20](#_Toc37)

[Список литературы 21](#_Toc38)

# Введение

Современное общество сталкивается с постоянным ростом угроз безопасности, что делает проблему защиты персональной информации, особенно учетных данных, особенно актуальной. В условиях повсеместного использования различных онлайн-сервисов (банков, социальных сетей, корпоративных систем) пользователи ежедневно вводят пароли, обеспечивающие доступ к их личной информации. Однако с ростом числа онлайн-аккаунтов возрастает и сложность безопасного хранения паролей. Одним из самых эффективных способов защиты паролей является их шифрование, что позволяет предотвратить утечку данных даже в случае компрометации базы данных.

Разработка менеджера паролей является важной задачей в контексте повышения безопасности пользователей в сети интернет. Программы такого рода позволяют не только организовать удобное и безопасное хранение паролей, но и предоставляют возможность пользователю выбрать наиболее подходящий способ шифрования данных. Таким образом, менеджеры паролей обеспечивают баланс между удобством работы с множеством учетных записей и высокой степенью защиты этих данных.

Целью данной курсовой работы является разработка менеджера паролей, который бы позволял пользователю надежно хранить учетные данные, предоставляя ему возможность выбора различных алгоритмов шифрования: Base64, MD5, SHA256, шифр Фейстеля, а также поддерживать механизм «соли», что обеспечивает дополнительную защиту паролей.

В данной работе будет рассмотрена структура и функциональность менеджера паролей, а также алгоритмы, которые могут быть использованы для реализации защиты данных. Особое внимание будет уделено разработке интерфейса пользователя, удобству работы с программой, а также безопасности хранения логинов и паролей.

В последние годы тема безопасности данных и защита личной информации становятся все более актуальными. С каждым днем увеличивается количество пользователей, использующих различные онлайн-сервисы, а вместе с этим растет и количество угроз, направленных на получение несанкционированного доступа к личным данным. В 2023 году было зафиксировано более 500 крупных утечек данных, что подтверждает острую необходимость в разработке эффективных решений для защиты учетных записей пользователей. Хищение паролей через утечку базы данных или фишинговые атаки — одна из самых распространенных форм киберугроз. Это требует поиска новых решений в области безопасного хранения и управления паролями.

Менеджеры паролей, как инструмент для решения проблемы безопасности, позволяют пользователю не только генерировать уникальные пароли для каждого аккаунта, но и хранить их в зашифрованном виде. При этом важным аспектом является выбор метода шифрования, поскольку разные алгоритмы имеют разные уровни защиты. Например, MD5, хоть и популярен, давно признан ненадежным, а более сложные алгоритмы, такие как SHA256 и шифры типа Фейстеля, обеспечивают гораздо более высокий уровень безопасности. Однако вряд ли один и тот же алгоритм может быть оптимальным решением для всех пользователей. Разработка универсального менеджера паролей, который позволяет выбирать метод шифрования, предоставляет пользователю возможность адаптировать систему под свои нужды.

Одним из центральных вопросов при разработке безопасной системы хранения паролей является выбор подходящего алгоритма шифрования. В контексте менеджера паролей важно, чтобы приложение обеспечивало надежное хранение данных, а также позволяло пользователю быть уверенным в том, что его учетные данные не будут скомпрометированы. Основная проблема заключается в выборе между различными алгоритмами шифрования, каждый из которых имеет свои преимущества и недостатки.

1. Base64 — это алгоритм, который не является полноценным методом шифрования, а скорее кодированием данных. Он часто используется для преобразования бинарных данных в текстовый формат. В его случае данные могут быть дешифрованы без потери информации, что делает этот метод не очень безопасным для хранения паролей, однако его использование может быть удобным для представления паролей в удобочитаемом виде.

2. MD5 — алгоритм хэширования, который раньше использовался для защиты паролей. Однако на сегодняшний день MD5 считается уязвимым к атакам с использованием радужных таблиц и перебора значений, что делает его небезопасным для хранения критической информации. Существуют быстрые методы взлома MD5-хэшей, что ограничивает его применение в современных системах безопасности.

3. SHA256— алгоритм хэширования, входящий в семейство SHA-2, который является гораздо более безопасным и эффективным в защите паролей. В отличие от MD5, SHA256 на данный момент считается довольно устойчивым к атакам и является одной из рекомендуемых технологий для хэширования паролей.

4. Шифр Фейстеля— это тип блочного шифрования, использующий структуру, построенную на принципе разделения данных на несколько частей и применения серии операций к ним. Шифр Фейстеля отличается высокой гибкостью и эффективностью, и его использование в менеджере паролей предоставляет более сложную и надежную защиту.

5. Соль (подсаливание пароля— это дополнительная мера защиты, при которой к паролю добавляется случайная строка, называемая солью, перед применением хэширования. Это позволяет значительно увеличить стойкость к атакам, поскольку при одинаковых паролях хэши будут различными благодаря добавлению соли. Также это помогает защитить от использования предварительно вычисленных радужных таблиц.

Для того чтобы обеспечить надежное и удобное хранение паролей в рамках данного проекта, необходимо решить несколько ключевых задач:

1. Реализация интерфейса пользователя: Разработать простой и интуитивно понятный интерфейс, позволяющий пользователю регистрироваться в системе, добавлять, изменять и удалять пароли, а также выбирать метод шифрования для хранения данных.

2. Обеспечение многоуровневой безопасности: Реализовать систему аутентификации, которая будет требовать ввода логина и пароля при входе в приложение, а также предусмотреть возможность использования различных алгоритмов шифрования для хранения паролей.

3. Реализация алгоритмов шифрования и хэширования: Разработать и интегрировать в приложение несколько алгоритмов шифрования, включая Base64, MD5, SHA256, шифр Фейстеля, а также обеспечить возможность работы с солью для повышения уровня безопасности.

4. Обеспечение надежного хранения ключей и конфиденциальности данных: Ключи для шифрования и дешифрования паролей должны быть защищены при помощи алгоритмов шифрования для обеспечения безопасной генерации, хранения и обработки пользовательских паролей для сайта

5. Тестирование и оптимизация: После реализации всех основных функций необходимо провести тестирование системы, выявить возможные уязвимости и оптимизировать приложение для повышения производительности и надежности.

# Основная часть

## 1. Теоретическая часть

Менеджеры паролей представляют собой программные решения, предназначенные для безопасного хранения и управления паролями пользователей, а также других конфиденциальных данных, таких как логины и личные идентификаторы. Современные угрозы безопасности, такие как фишинг, утечка данных и несанкционированный доступ к учетным записям, требуют особого внимания к методам защиты паролей и их безопасному хранению.

Целью данной курсовой работы является разработка приложения-менеджера паролей, которое будет реализовывать базовые функциональные возможности по защите паролей с использованием алгоритмов шифрования и хэширования, таких как Base64, MD5, SHA256, и шифра Фейстеля, а также внедрять возможность использования соли для повышения уровня безопасности.

## 2. Постановка задачи

В процессе разработки данного приложения необходимо уделить внимание созданию интерфейса, который будет максимально удобен и интуитивно понятен для конечных пользователей. Важным этапом является также реализация системы регистрации, которая позволит пользователям создавать уникальные учетные записи и входить в систему с помощью логина и пароля. Каждый пользователь должен иметь возможность безопасно сохранять свои личные данные в зашифрованном виде, что обеспечит высокий уровень конфиденциальности и защиту от несанкционированного доступа.

Одной из ключевых задач является выбор и внедрение методов шифрования, которые будут использоваться для защиты данных. Приложение должно предоставлять пользователю несколько вариантов алгоритмов шифрования, среди которых он сможет выбрать наиболее подходящий для его нужд. Для повышения уровня безопасности в приложении должен быть реализован механизм выбора алгоритма шифрования, который поддерживает работу как с простыми методами, такими как Base64 и MD5, так и с более сложными, например, шифром Фейстеля и методами шифрования с солью.

Процесс шифрования и дешифрования данных должен быть понятен и доступен пользователю, чтобы даже человек, не обладающий глубокими знаниями в области информационной безопасности, мог эффективно использовать приложение для защиты своих данных. Важным аспектом является обеспечение надежной защиты данных, чтобы исключить возможность их взлома или утраты.

Одним из элементов, который значительно повышает уровень безопасности, является использование соли. Соль — это случайная строка данных, которая добавляется к паролю или другому важному элементу перед его хешированием. Это помогает предотвратить атаки радужных таблиц, которые часто применяются для быстрого взлома паролей. Внедрение этого механизма в приложение является необходимостью, если мы хотим обеспечить высокий уровень безопасности данных пользователей.

Кроме того, важно, чтобы пользователь мог легко и удобно взаимодействовать с приложением, не сталкиваясь с лишними сложностями. Интерфейс должен быть дружелюбным и понятным, а все операции с шифрованием должны выполняться быстро и без ошибок. Пользователь должен видеть четкие инструкции, а также иметь возможность выбрать алгоритм шифрования в зависимости от того, что ему требуется в данный момент.

В рамках данной разработки была поставлена задача реализации приложения, которое позволит пользователям выбрать один из четырех предложенных методов шифрования: Base64, MD5, шифр Фейстеля и Base64 с солью. Каждый из этих методов имеет свои особенности и области применения. Например, Base64 часто используется для кодирования данных, которые необходимо передавать по сети, так как этот алгоритм превращает бинарные данные в текстовый формат. Однако он не является методом шифрования и не гарантирует высокой степени безопасности.

Base64 — стандарт кодирования двоичных данных при помощи только 64 символов ASCII. Алфавит кодирования содержит латинские символы A-Z, a-z, цифры 0-9 (всего 62 знака) и 2 дополнительных символа, зависящих от системы реализации. Каждые 3 исходных байта кодируются четырьмя символами (увеличение на ¹⁄₃). Для того чтобы преобразовать данные в Base64, первый байт помещается в самые старшие восемь битов 24-битного буфера, следующий — в средние восемь и третий — в младшие восемь битов. Если кодируется менее чем три байта, то соответствующие биты буфера устанавливаются в ноль. Далее каждые шесть битов буфера, начиная с самых старших, используются как индексы строки «ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/», и её символы, на которые указывают индексы, помещаются в выходную строку. Если кодируется только один или два байта, в результате получаются только первые два или три символа строки, а выходная строка дополняется двумя или одним знаком =. Это предотвращает добавление дополнительных битов к восстановленным данным. Процесс повторяется над оставшимися входными данными. При кодировании Base64 размер сообщения увеличивается приблизительно на 33 %. Это надо учитывать, если есть ограничения на размер конечного сообщения. Так, при максимально допустимом размере 64 МБ реальный размер передаваемого сообщения должен быть не более 48 МБ [8].   
Хэш-функция предназначена для свертки входного массива любого размера в битовую строку, для MD5 длина выходной строки равна 128 битам.  
Чаще всего хэш-функции используются для проверки уникальности пароля, файла, строки и тд. К примеру, скачивая файл из интернета, вы часто видите рядом с ним строку вида b10a8db164e0754105b7a99be72e3fe5 — это и есть хэш, прогнав этот файл через алгоритм MD5 вы получите такую строку, и, если хэши равны, можно с большой вероятностью утверждать что этот файл действительно подлинный.

Сеть Фейстеля, или конструкция Фейстеля (англ. Feistel network, Feistel cipher), — один из методов построения блочных шифров. Сеть состоит из ячеек, называемых ячейками Фейстеля. На вход каждой ячейки поступают данные и ключ. На выходе каждой ячейки получают изменённые данные и изменённый ключ. Все ячейки однотипны, и говорят, что сеть представляет собой определённую многократно повторяющуюся (итерированную) структуру. Ключ выбирается в зависимости от алгоритма шифрования/расшифрования и меняется при переходе от одной ячейки к другой. При шифровании и расшифровании выполняются одни и те же операции; отличается только порядок ключей. Ввиду простоты операций сеть Фейстеля легко реализовать как программно, так и аппаратно. Ряд блочных шифров (DES, RC2, RC5, RC6, Blowfish, FEAL, CAST-128, TEA, XTEA, XXTEA и др.) использует сеть Фейстеля в качестве основы. Альтернативой сети Фейстеля является подстановочно-перестановочная сеть (AES и др.). В 1971 году Хорст Фейстель запатентовал два устройства, реализующие различные алгоритмы шифрования, позже получившие название «Люцифер» (англ. Lucifer). Одно из этих устройств использовало конструкцию, впоследствии названную «сетью Фейстеля» (англ. Feistel cipher, Feistel network). Тогда Фейстель работал над созданием новых криптосистем в стенах IBM вместе с Доном Копперсмитом. Проект «Люцифер» был скорее экспериментальным, но стал основой для алгоритма DES (англ. data encryption standard). В 1973 году журнал «Scientific American» опубликовал статью Фейстеля «Криптография и компьютерная безопасность» (англ. Cryptography and computer privacy), в которой раскрыты некоторые важные аспекты шифрования и приведено описание первой версии проекта «Люцифер». В первой версии проекта «Люцифер» сеть Фейстеля не использовалась.

На основе сети Фейстеля был спроектирован алгоритм DES. В 1977 году власти США приняли стандарт FIPS 46-3, признающий DES стандартным методом шифрования данных. DES некоторое время широко использовался в криптографических системах. Итеративная структура алгоритма позволяла создавать простые программные и аппаратные реализации.

Согласно некоторым данным, в СССР уже в 1970-е годы КГБ разрабатывала блочный шифр, использовавший сеть Фейстеля, и, вероятно, именно этот шифр в 1990 году был принят в качестве ГОСТ 28147-89.

В 1987 году были разработаны алгоритмы FEAL и RC2. Сети Фейстеля получили широкое распространение в 1990-е годы — в годы появления таких алгоритмов, как Blowfish (1993), TEA (1994), RC5 (1994), CAST-128 (1996), XTEA (1997), XXTEA (1998), RC6 (1998) и других.

2 января 1997 года институт NIST объявил конкурс по созданию нового алгоритма шифрования данных, призванного заменить DES. Новый блочный шифр получил название AES (англ. advanced encryption standard) и был утверждён 26 мая 2002 года. В AES вместо сети Фейстеля используется подстановочно-перестановочная сеть.

Метод Base64 с солью представляет собой сочетание двух технологий, каждая из которых играет свою важную роль в обеспечении безопасности данных. В первую очередь, соль используется для того, чтобы сделать зашифрованные данные уникальными, а кодирование в Base64 позволяет эффективно передавать эти данные через текстовые каналы. Эта комбинация помогает значительно повысить уровень защиты информации, предотвращая атаки, направленные на расшифровку или восстановление исходных данных, и делает процесс передачи данных более безопасным и удобным.

Соль в криптографии выполняет функцию случайных данных, которые добавляются к исходным данным перед их шифрованием или хешированием. Основная цель соли — предотвратить использование предварительно вычисленных значений, таких как радужные таблицы, и усложнить атакующим процесс расшифровки данных. Если бы соль не использовалась, то одинаковые входные данные (например, одинаковые пароли) всегда приводили бы к одинаковым результатам, что могло бы быть использовано для взлома. Однако добавление соли делает каждый процесс шифрования уникальным, даже если два пользователя выбирают идентичные пароли. Таким образом, соль эффективно защищает данные от стандартных атак, таких как атаки по словарю, где злоумышленники используют заранее подготовленные списки популярных паролей и хешей для быстрого подбора.

Кроме того, важной частью этого метода является кодирование данных в Base64. Алгоритм Base64 преобразует бинарные данные в строку, которая состоит только из безопасных символов, подходящих для передачи через текстовые каналы. Это особенно важно в тех случаях, когда данные необходимо передавать через протоколы, которые не поддерживают работу с бинарными данными, такие как электронная почта или HTTP-запросы. Используя Base64, зашифрованные данные становятся пригодными для передачи и хранения, что делает этот метод идеальным для защиты информации в интернете и других системах, где необходима совместимость с текстовыми данными.

Комбинированный метод Base64 с солью представляет собой несколько этапов обработки данных. На первом шаге генерируется соль, которая будет уникальной для каждого пользователя или каждого процесса шифрования. Эта соль добавляется к исходным данным, будь то пароль или любая другая информация, что приводит к изменению исходной строки. После этого, данные подвергаются шифрованию с использованием выбранного криптографического алгоритма. В зависимости от потребностей системы, это может быть как хеширование, так и симметричное или асимметричное шифрование. В конечном итоге, результат шифрования кодируется в формат Base64, превращая его в строку, которая может быть безопасно передана через любые текстовые каналы, такие как электронная почта, веб-запросы и другие. Это обеспечивает удобство работы с зашифрованными данными и их безопасную передачу.

Одним из главных преимуществ использования соли является то, что она значительно усложняет атаки, направленные на расшифровку данных. Даже если злоумышленник получит доступ к базе данных, содержащей зашифрованную информацию, без знания соли ему будет крайне сложно восстановить исходные данные. Это особенно важно в случае хранения паролей пользователей или других конфиденциальных данных, которые могут быть потенциально украдены. Соль помогает предотвратить использование предсчитанных хешей для подбора паролей, а также делает невозможным использование радужных таблиц, которые могут быть эффективно применены для стандартных, незащищенных данных.

В свою очередь, использование Base64 делает данные удобными для передачи по сетям и в текстовых протоколах. Стандартные бинарные данные, такие как результаты криптографических операций, могут быть трудны для обработки или передачи через некоторые системы, которые не поддерживают работу с бинарным кодом. В таких случаях кодирование в Base64 позволяет преобразовать результат шифрования в строку, которая легко воспринимается большинством протоколов и приложений. Это делает метод Base64 с солью идеальным для применения в веб-разработке, а также для интеграции с различными API и веб-сервисами, где необходимо безопасно передавать данные в текстовом виде.

Процесс применения Base64 с солью можно представить следующим образом: сначала генерируется случайная соль, которая добавляется к исходным данным. После этого данные шифруются с использованием соответствующего криптографического алгоритма, и результат преобразуется в строку формата Base64. Этот процесс может быть применен не только к паролям, но и к другим конфиденциальным данным, таким как токены доступа, личные данные пользователей или любые другие данные, которые требуют дополнительной защиты. Например, если пользователь вводит свой пароль, система генерирует соль, добавляет её к паролю, а затем шифрует полученную строку. Результат кодируется в Base64 и сохраняется в базе данных или передается через сеть. Когда пользователь снова пытается войти в систему, процесс повторяется, и сравнение зашифрованных значений позволяет системе проверить корректность введенного пароля.

Этот метод позволяет обеспечить высокий уровень безопасности даже в условиях, когда база данных может быть скомпрометирована. В отличие от простого хранения паролей в виде хешей, добавление соли делает невозможным использование радужных таблиц для дешифровки данных, что значительно повышает устойчивость системы к атакам. Это особенно актуально в современных веб-приложениях, где информация о пользователях может быть уязвимой для кражи или утечки.

Таким образом, комбинация соли и Base64 представляет собой эффективное средство защиты данных, которое сочетает в себе два важных аспекта безопасности: уникализацию данных с помощью соли и удобство передачи зашифрованных данных с использованием Base64. Этот метод является важным инструментом для предотвращения атак на пароли, токены и другие конфиденциальные данные, а также для обеспечения их безопасной передачи через различные каналы связи.

Одной из ключевых задач при разработке этого приложения является обеспечение того, чтобы каждый из предложенных методов шифрования был реализован максимально эффективно и безопасно. Важно, чтобы алгоритмы шифрования были адаптированы для работы в реальном времени и не приводили к значительному замедлению работы приложения. Для этого разработчики должны учесть различные оптимизации, такие как использование современных библиотек для работы с криптографией и обеспечение правильной обработки ошибок.

Также, помимо реализации шифрования, необходимо создать механизм для хранения данных в зашифрованном виде. Данные должны сохраняться в базе данных, при этом доступ к ним должен быть возможен только при вводе правильного пароля или при использовании других методов аутентификации. Хранение паролей и других конфиденциальных данных в зашифрованном виде является обязательным условием для обеспечения безопасности приложения.

Особое внимание стоит уделить тестированию приложения на безопасность. Для этого необходимо провести серию тестов, направленных на выявление уязвимостей, таких как возможность взлома пароля с помощью радужных таблиц, атаки на слабые алгоритмы шифрования или на ошибки, связанные с обработкой данных. Это позволит не только удостовериться в безопасности приложения, но и повысить доверие пользователей к использованию этого программного продукта.

Приложение должно обеспечивать надежную защиту данных с помощью различных криптографических методов, включая соль, и при этом оставаться удобным и доступным для всех категорий пользователей.

## 3. Архитектура и структура приложения

Разработанное приложение реализовано с использованием языка программирования Java и библиотеки для создания графических интерфейсов (GUI) \*\*Swing\*\*. Основной компонент программы представляет собой окно с полями для ввода логина и пароля, выпадающим списком для выбора метода шифрования и кнопкой для подтверждения ввода данных.

### 3.1. Структура проекта

Проект включает несколько классов, каждый из которых отвечает за отдельный аспект работы программы:

- PasswordManagerApp — основной класс, отвечающий за создание и отображение графического интерфейса, а также за логику шифрования и хэширования паролей.

- PasswordGeneratorMenu — класс, который открывается после успешного ввода данных, предлагая пользователю функционал для генерации паролей с использованием различных параметров.

- Logger log4j2 — используется для ведения логирования событий, ошибок и действий пользователя, что позволяет отслеживать работу приложения и диагностировать проблемы.

### 3.2. Графический интерфейс

Для создания графического интерфейса используется стандартная библиотека Java Swing которая предоставляет набор инструментов для создания оконных приложений. В интерфейсе предусмотрены следующие элементы:

- JTextField для ввода логина пользователя.

- JPasswordField для ввода пароля пользователя.

- JComboBox для выбора метода шифрования.

- JButton для подтверждения ввода.

- JCheckBox для опции генерации паролей с случайными параметрами.

Пример интерфейса:

1. В верхней части экрана располагаются поля для ввода логина и пароля.

2. В центре экрана — выпадающий список для выбора метода шифрования.

3. В нижней части — кнопка для выполнения входа.

4. Реализация алгоритмов шифрования и хэширования

Одной из ключевых особенностей приложения является возможность выбора различных методов шифрования паролей, что позволяет пользователю выбрать наилучший способ защиты своих данных. Для реализации этих методов в программе используются стандартные библиотеки Java, а также алгоритмы, разработанные для работы с конкретными криптографическими задачами.

### 3.3 Блок-схема приложения

#### 3.3.1 Блок-схема алгоритма

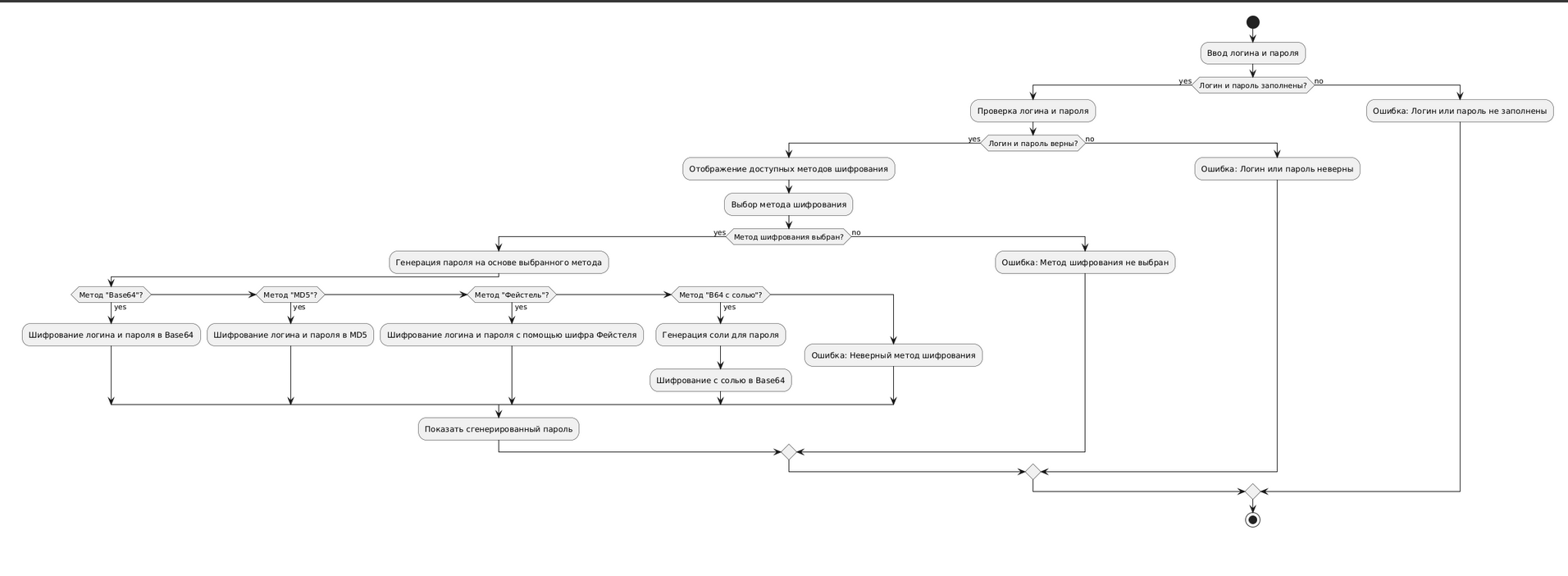


Рисунок 1 – блок-схема алгоритма

#### 3.3.1 Блок-схема методов

Метод конструктора PasswordGeneratorMenu (из PasswordGeneratorMenu.java)

В конструкторе создаются элементы GUI, такие как текстовые поля и кнопки. Затем, при нажатии кнопки "Получить", происходит генерация пароля в зависимости от выбранной опции (useRandomPassword).

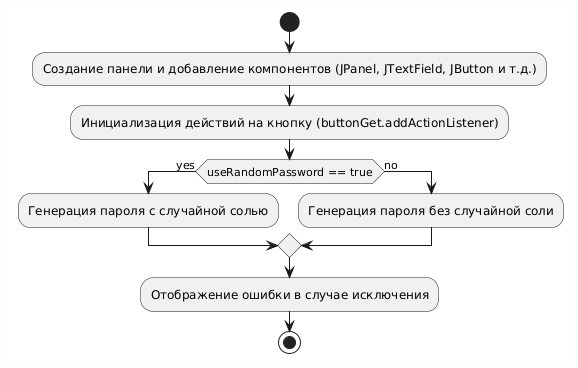


Рисунок 2 – блок-схема метода PasswordGeneratorMenu

Метод genPass (из PasswordGeneratorMenu.java)

Этот метод генерирует пароль для сайта, комбинируя логин, сайт и пароль, и хэширует результат с помощью алгоритма SHA-256.



Рисунок 3 – блок-схема метода genPass

Метод generateRandomSalt (из PasswordGeneratorMenu.java)

Метод генерирует случайную строку заданной длины, используя символы от 'a' до 'z'. Это используется для генерации соли.



Рисунок 4 – блок-схема метода generateRandomSalt

Метод конструктора PasswordManagerApp (из PasswordManagerApp.java):

В конструкторе создаются основные элементы GUI для входа в систему и выбора метода шифрования. Также добавляется слушатель на кнопку "Войти", который выполняет действия при ее нажатии.



Рисунок 5 – блок-схема метода PasswordManagerApp

Методы encryptWithSalt, encryptMD5, encryptFeistel, getSHA256Hash принимают на вход строку и возвращают ее в зашифрованном виде



Рисунок 6 – блок-схема методов шифрования

3.4 UML-диаграммы

#### 3.4.1 Диаграмма классов

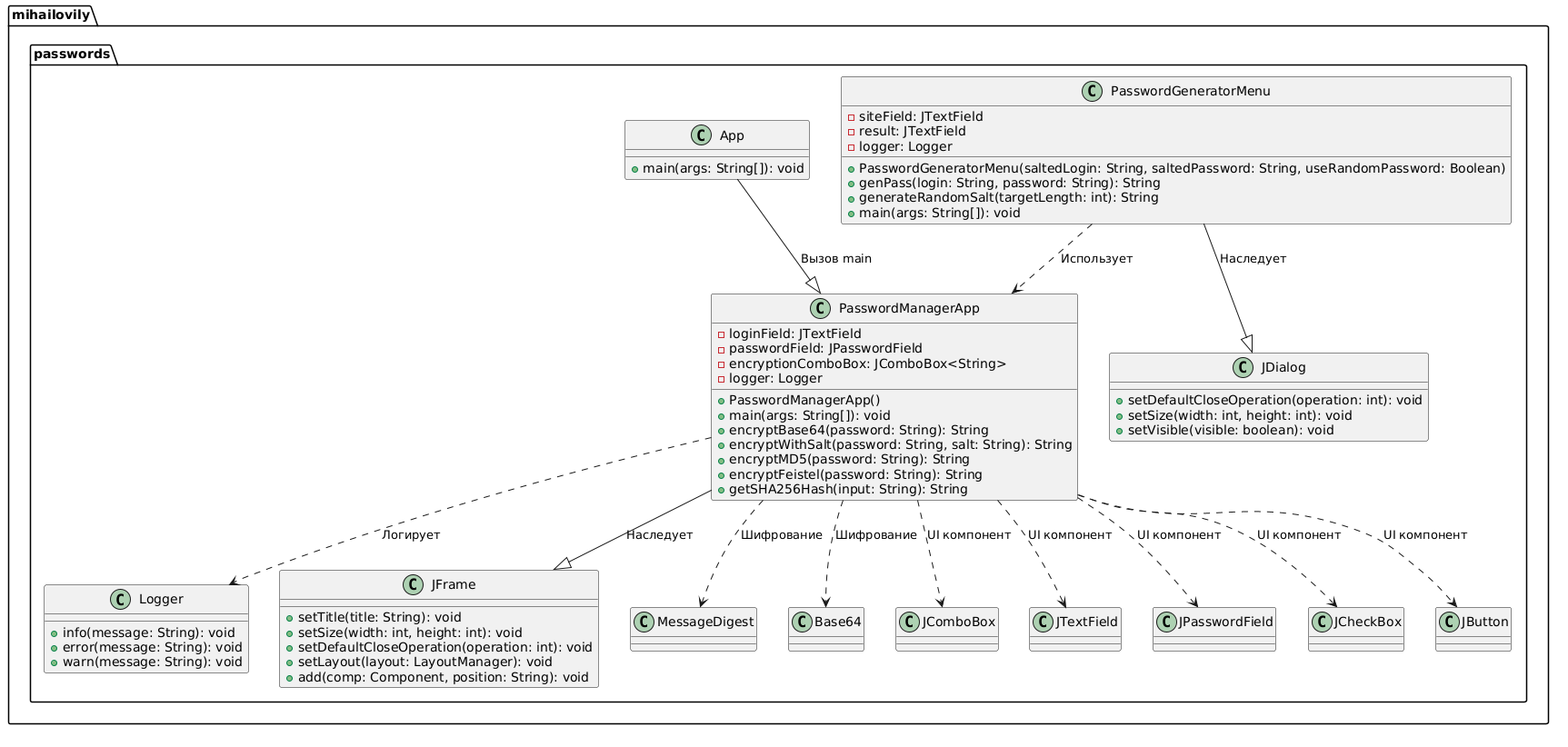


Рисунок 7 – диаграмма классов

#### 3.4.2 Диаграмма прецедентов

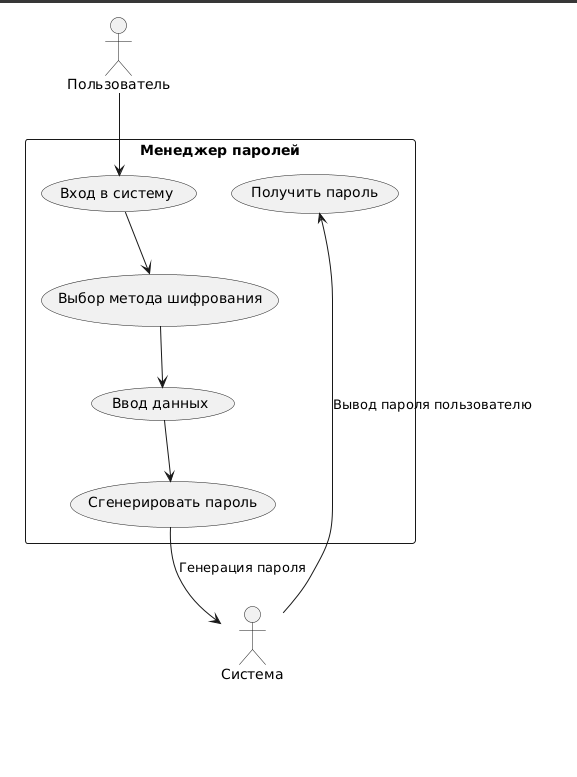


Рисунок 8 – диаграмма прецедентов

### 3.5 Положительные и негативные примеры работы приложения

#### 3.5.1 Положительные примеры

#### 3.5.2 Негативные примеры

### 3.6 Библиотеки, сторонние классы и интерфейсы, которые используются в коде

Библиотеки (Libraries)

Библиотека в Java представляет собой набор заранее подготовленных классов и методов, которые можно использовать для решения различных задач. Библиотеки предлагают готовые решения для общих проблем, таких как сетевые взаимодействия, обработка данных, создание графических интерфейсов и другие.

Например, библиотека java.util включает коллекции, такие как списки, множества и карты.

Сторонние классы (Third-party Classes)

Сторонние классы — это классы, которые разрабатываются и поддерживаются независимыми разработчиками или организациями, а не являются частью стандартной библиотеки Java.

Примеры таких классов включают различные библиотеки и фреймворки, такие как DoubleSummaryStatistics, Scanner и другие, которые предоставляют функциональность для работы с файлами, базами данных, упрощают процесс разработки и решают другие задачи.

Интерфейсы (Interfaces)

В Java интерфейс — это абстрактный тип данных, который определяет набор методов, которые должны реализовать классы. Интерфейсы устанавливают контракт, определяя, какие методы должны быть реализованы, но сами по себе не содержат реализации этих методов.

В разработанном приложении используются следующие сторонние библиотеки:

#### 3.6.1. javax.swing.\* (Java Swing)

JFrame: Это основа для создания графического пользовательского интерфейса (GUI) в Java. Класс используется для создания основного окна приложения.

JDialog: Класс для создания модальных и немодальных окон в Java. В вашем коде используется для создания окна генератора паролей.

JTextField: Используется для создания текстовых полей, в которые пользователь может вводить текст (например, для логина или пароля).

JPasswordField: Это разновидность JTextField, предназначенная для ввода пароля. Она скрывает вводимые символы.

JComboBox: Компонент для создания выпадающего списка, используемого для выбора метода шифрования.

JCheckBox: Компонент для создания флажков (чекбоксов), который используется для выбора опции генерации пароля с случайной солью.

JButton: Класс для создания кнопок, которые вызывают действия при нажатии.

JPanel: Контейнер для других компонентов GUI, используемый для размещения элементов интерфейса.

JLabel: Компонент для отображения текста или изображений в интерфейсе.

WindowAdapter, WindowEvent: Эти классы используются для обработки событий окна, таких как его закрытие.

Роль: Библиотека Swing предоставляет все необходимые компоненты для создания и обработки GUI, взаимодействия с пользователем и отображения информации.

#### 3.6.2. org.apache.logging.log4j.\* (Apache Log4j)

Logger: Это главный интерфейс для записи логов. В коде используется для ведения логов различных операций в приложении (например, при генерации пароля, при ошибках и т.д.).

LogManager: Это фабричный класс, который создает объект Logger. Используется для получения логгера для текущего класса.

Роль: Log4j — это мощная библиотека для ведения логирования в Java-приложениях. В коде используется для отслеживания и записи событий (например, ошибок, генерации паролей и успешных операций).

3.6.3. java.awt.\* (Abstract Window Toolkit)

BorderLayout: Используется для управления расположением компонентов в контейнере. Он делит контейнер на пять областей: NORTH, SOUTH, EAST, WEST и CENTER.

GridLayout: Позволяет располагать компоненты в сетке с фиксированным числом строк и столбцов.

Font: Класс для работы с шрифтами в интерфейсе.

Color: Класс для работы с цветами в интерфейсе (не используется в вашем коде, но доступен через AWT).

Роль: Библиотека AWT предоставляет основные средства для работы с пользовательским интерфейсом, такими как размещение компонентов на экране и управление их размерами.

#### 3.6.4. java.security.\* (Java Security API)

MessageDigest: Класс для создания хеш-значений с использованием различных криптографических алгоритмов, таких как SHA-256 и MD5. Используется для хеширования логина, пароля и других данных.

NoSuchAlgorithmException: Исключение, которое выбрасывается, если запрашиваемый криптографический алгоритм не существует (например, если попытаться получить хеш с использованием алгоритма, которого нет в библиотеке).

Роль: Библиотека Java Security предоставляет классы для работы с криптографией, включая хеширование и шифрование. В вашем коде используется для хеширования паролей и логинов с использованием алгоритмов SHA-256 и MD5.

#### 3.6.5. java.util.\* (Java Utilities)

Base64: Класс для кодирования и декодирования данных в формате Base64. В коде используется для шифрования паролей и логинов в формат Base64.

Random: Класс для генерации случайных чисел. В коде используется для создания случайной соли.

StringBuilder: Класс для эффективного построения строк в Java. В коде используется для создания строки случайной соли и преобразования данных в строки.

Arrays: Класс с методами для работы с массивами (например, копирование данных в массивы). В коде используется для копирования данных в массивы, например, при работе с массивами байтов.

Роль: Библиотека java.util предоставляет полезные утилиты для работы с коллекциями, строками, случайными числами и другими вспомогательными функциями. В вашем коде она используется для генерации случайных данных, работы с строками и кодирования.

#### 3.6.6. java.awt.event.\* (Java AWT Event Handling)

WindowAdapter: Класс, который позволяет обрабатывать события окна. В коде используется для обработки закрытия окна.

WindowEvent: Класс, представляющий событие, происходящее при изменении состояния окна, например, при его закрытии.

Роль: Эти классы используются для обработки событий, связанных с окнами (например, когда окно закрывается). В вашем коде WindowAdapter позволяет закрыть окно приложения при завершении работы.

#### 3.6.7. java.lang.\* (Java Core)

String: Класс для представления строк в Java.

System: Класс для взаимодействия с системными ресурсами. Например, используется для передачи аргументов в программу.

Integer: Класс-обертка для работы с типом int.

RuntimeException: Исключение, которое выбрасывается, если происходит необработанная ошибка в процессе выполнения программы.

Роль: Классы из пакета java.lang являются основными классами Java и используются для обработки строк, работы с примитивными типами данных, исключениями и взаимодействия с системой.

#### 3.6.8. javax.swing.event.\* (Swing Event Handling)

ActionListener: Интерфейс для обработки событий, возникающих при взаимодействии с компонентами, такими как кнопки или текстовые поля. В коде используется для обработки нажатий кнопок, например, при нажатии на кнопку "Войти" или "Получить".

Роль: Интерфейс для обработки действий, связанных с элементами пользовательского интерфейса, такими как кнопки и другие компоненты.

Вывод

Swing: используется для создания графического интерфейса пользователя (GUI).

Log4j: используется для ведения логирования и отслеживания событий.

AWT: предоставляет инструменты для компоновки и работы с графическими компонентами.

Java Security API: используется для хеширования и шифрования данных.

Java Utilities: предоставляет полезные утилиты для работы с коллекциями, строками и случайными данными.

Java Event Handling: используется для обработки событий в приложении.

Каждая из этих библиотек играет свою роль в реализации функциональности приложения и организации его работы.

## 4. Реализация шифрования

### 4.1. Шифрование Base64

Base64 — это метод кодирования данных в строку ASCII, который используется для представления бинарных данных в текстовом формате. Он не является методом шифрования, а скорее методом кодирования, который позволяет передавать бинарные данные, такие как пароли, через текстовые каналы.

private String encryptBase64(String password) {

return Base64.getEncoder().encodeToString(password.getBytes());

}

В данном методе строка пароля конвертируется в байтовый массив, а затем происходит кодирование с помощью Base64. Результатом является строка.

### 4.2. Хэширование MD5

MD5 (Message Digest Algorithm 5) — это криптографический хэш-функция, которая используется для создания фиксированной длины хеша из входных данных. Несмотря на свою популярность в прошлом, MD5 сегодня считается уязвимым к атакам, и его использование не рекомендуется для защиты паролей. Однако в рамках данного приложения он был выбран для демонстрации старых методов защиты.

private String encryptMD5(String password) throws NoSuchAlgorithmException {

MessageDigest md = MessageDigest.getInstance("MD5");

byte[] hash = md.digest(password.getBytes());

StringBuilder hexString = new StringBuilder();

for (byte b : hash) {

hexString.append(Integer.toHexString(0xFF & b));

}

return hexString.toString();

}

Метод MD5 применяет хэш-функцию, которая преобразует исходный пароль в строку фиксированной длины. Результат хэширования в данном случае — это строка в шестнадцатеричном формате.

### 4.3. Шифр Фейстеля

Шифр Фейстеля — это блоковый шифр, который используется в криптографических системах. Он основан на принципе разделения данных на две половины и применении несколько раундов преобразований.

private String encryptFeistel(String password) {

return feistelCipher(password.getBytes());

}

Метод feistelCipher принимает байтовый массив, разделяет его на две части и выполняет несколько раундов операций XOR для их модификации.

### 4.4. Шифрование с солью (Salted Base64)

Использование соли является важной техникой защиты паролей. Соль представляет собой случайно сгенерированное значение, которое добавляется к паролю перед его хэшированием или шифрованием. Это помогает предотвратить использование радужных таблиц и усиливает защиту.

private String encryptWithSalt(String password, String salt) {

String saltedPassword = password + salt;

return Base64.getEncoder().encodeToString(saltedPassword.getBytes());

}

В этом методе соль добавляется к паролю, и результат кодируется в Base64. При такой реализации даже одинаковые пароли будут иметь разные хеши, если соль отличается.

## 5. Обработка ошибок и логирование при помощи log4j

Важной частью любого приложения является обработка ошибок и ведение логов. В рамках данной работы для ведения логов используется библиотека Log4j2, которая позволяет фиксировать важные события в приложении, такие как ошибки, предупреждения и информацию о действиях пользователя. Логирование позволяет отслеживать, что происходит в приложении в определенный момент времени. Это полезно для мониторинга выполнения программы и выявления проблем.

При возникновении ошибок или непредвиденных ситуаций логирование может предоставить дополнительную информацию для диагностики и выявления причин проблем.

Запись логов также может быть использована для мониторинга производительности приложения, выявления узких мест и оптимизации кода.

Логирование позволяет создавать отчеты о работе приложения, которые могут быть использованы для аудита или в случае необходимости отчетности.

**Пример использования логирования:**

private static final Logger logger = LogManager.getLogger(PasswordManagerApp.class);

logger.info("Приложение запущено.");

logger.warn("Поля логина и пароля не должны быть пустыми.");

logger.error("Произошла ошибка при обработке пароля: ", ex);

## 6. Сборка gradle

После реализации алгоритмов и интерфейса было проведено тестирование приложения для проверки его работоспособности и выявления возможных ошибок. Особое внимание было уделено тестированию различных методов шифрования и проверки их корректности.

Gradle - это система сборки (build tool) для автоматизации процесса построения проектов в различных языках программирования, включая Java, Kotlin, Groovy и другие. Gradle позволяет управлять зависимостями, компилировать исходный код, выполнять тестирование, собирать и упаковывать приложения, а также выполнять множество других задач, необходимых для разработки программного обеспечения.

Для запуска приложения неободимо ввести команду gradle run в консоли в папке проекта. Для сборки приложения в виде jar необходима команда gradle jar.

# Заключение

Разработка менеджера паролей, представленного в рамках данной курсовой работы, является важным шагом в решении одной из актуальных задач современного программирования — обеспечения безопасности и защиты пользовательских данных. В условиях постоянного роста угроз безопасности, особенно в контексте использования различных онлайн-сервисов, защита персональной информации становится необходимостью, а средства для надежного хранения паролей и другой конфиденциальной информации становятся неотъемлемой частью повседневной жизни пользователей интернета.

Менеджеры паролей, представляющие собой специальные программы для безопасного хранения и управления паролями, позволяют пользователю эффективно решать проблемы безопасности, предотвращать утечку данных и защищать свои учетные записи от несанкционированного доступа. Важно отметить, что использование таких программ помогает избежать ряда человеческих ошибок, таких как использование одинаковых паролей для разных сервисов, что является одной из самых распространенных причин утечек данных в сети.

Суть разработки менеджера паролей в рамках данной работы заключается в создании программы, которая обеспечивала бы пользователю возможность не только удобного хранения паролей, но и их надежного шифрования. Реализованные в программе алгоритмы шифрования, такие как Base64, MD5, SHA256, а также более сложный и защищенный шифр Фейстеля, позволяют выбрать наиболее подходящий вариант защиты в зависимости от предпочтений пользователя. Более того, внедрение функции «соли» предоставляет дополнительный уровень безопасности, предотвращая атаки с использованием радужных таблиц и усиливая стойкость хэширования.

Реализация менеджера паролей в рамках курсовой работы показала, что создание приложения, которое сочетает в себе простоту использования и высокий уровень безопасности, является вполне осуществимой задачей. В процессе работы над проектом был проделан значительный объем работы, включающий не только выбор и реализацию алгоритмов шифрования, но и проработку интерфейса, создание системы аутентификации и организацию надежного хранения ключей и паролей.

Одним из важнейших этапов разработки стала реализация многоуровневой системы безопасности, включающей регистрацию пользователя, шифрование паролей, а также выбор различных методов шифрования. Важно, что эти алгоритмы, несмотря на свою разную сложность и степень безопасности, были интегрированы в одну систему, что позволило пользователю выбрать наиболее удобный и надежный способ защиты. Каждому методу шифрования были уделены должное внимание и подробное тестирование, что позволило убедиться в их эффективности и устойчивости к потенциальным атакам.

Особое внимание было уделено реализации интерфейса программы. Удобство использования менеджера паролей имеет важное значение для обеспечения его популярности и практическости. Интерфейс должен быть интуитивно понятным, доступным для любого пользователя, не обладающего глубокими знаниями в области криптографии или программирования. В этом контексте создание простой и понятной формы для ввода логина и пароля, возможность выбора алгоритма шифрования и управления паролями являются неотъемлемой частью приложения. Все эти элементы были реализованы с учетом удобства и функциональности, чтобы обеспечить пользователю комфорт при работе с программой.

Кроме того, внимание было уделено не только функциональности приложения, но и безопасности хранения ключей и паролей. Для этого были использованы подходы, обеспечивающие хранение конфиденциальных данных в защищенном виде, что является необходимым для создания по-настоящему безопасного менеджера паролей. Это позволило минимизировать риски утечек данных и повысить уровень доверия пользователей к приложению.

Несмотря на достигнутые результаты, в ходе разработки приложения были выявлены несколько ограничений и проблем, которые в будущем могут быть улучшены или оптимизированы. Одним из таких ограничений является использование устаревших алгоритмов, таких как MD5, который на сегодняшний день считается небезопасным для криптографической защиты паролей. В реальной практике рекомендуется использовать более современные и стойкие алгоритмы, такие как SHA-256 или алгоритмы, основанные на криптографических методах с солью. Однако, несмотря на это, MD5 был включен в проект с целью продемонстрировать, как исторически устаревшие методы шифрования могут быть адаптированы в современных приложениях с учетом их недостатков.

Другим ограничением является необходимость дальнейшего повышения безопасности хранения и обработки ключей. Несмотря на внедрение защищенного хранения ключей, в идеале требуется использование более сложных методов защиты, таких как аппаратные модули безопасности (HSM) или использование многофакторной аутентификации для защиты ключей доступа. Эти аспекты являются важными в контексте корпоративных решений и для приложений, работающих с высокочувствительными данными.

Еще одним вопросом, который требует дальнейшего внимания, является возможность интеграции с внешними сервисами и приложениями для автоматической генерации паролей, а также добавление функций для двухфакторной аутентификации. Включение таких функций значительно повысило бы уровень безопасности и удобства использования программы, а также расширило бы ее функциональные возможности.

Для повышения функциональности и безопасности данного менеджера паролей можно предложить несколько направлений для дальнейшей работы. Во-первых, стоит рассмотреть возможность использования более современных и безопасных алгоритмов шифрования и хэширования. Например, алгоритмы, такие как bcrypt или Argon2, предлагают дополнительный уровень защиты за счет встроенной сложности обработки, что делает их более эффективными для защиты паролей от атак с использованием современных вычислительных мощностей.

Кроме того, полезным улучшением было бы добавление механизма восстановления доступа к учетной записи через использование резервных ключей или многофакторной аутентификации. Это могло бы значительно повысить уровень безопасности приложения, а также уменьшить риски, связанные с потерей пароля или утратой доступа.

Другим важным направлением для улучшения является улучшение производительности программы, особенно при обработке большого количества паролей и данных. Это можно достичь путем оптимизации алгоритмов шифрования и хэширования, а также улучшения архитектуры приложения с точки зрения многозадачности и параллельных вычислений.

Кроме того, в будущем можно было бы рассмотреть интеграцию с различными облачными сервисами, что позволило бы пользователям хранить свои пароли в защищенном облачном хранилище и получать доступ к своим данным с различных устройств. Однако для этого потребуется значительно усилить систему защиты, поскольку облачные хранилища могут подвергаться дополнительным угрозам безопасности.

В итоге, создание менеджера паролей в рамках данной курсовой работы стало важным и полезным опытом как в плане теоретических знаний о криптографии и алгоритмах шифрования, так и в плане практических навыков разработки программного обеспечения, ориентированного на безопасность. Программа, разработанная в ходе работы, является примером того, как можно эффективно решать проблемы безопасности при работе с конфиденциальной информацией, и может служить основой для более сложных и многофункциональных решений в области защиты данных.

Менеджеры паролей продолжают оставаться важным инструментом в области информационной безопасности, и их развитие будет актуальным на протяжении многих лет. Все эти разработки и улучшения направлены на обеспечение того, чтобы пользователь мог безопасно хранить и использовать свои пароли без риска для личной информации, а также мог выбрать наиболее подходящий метод защиты, исходя из своих потребностей и предпочтений. В конечном итоге, создание более безопасных и удобных инструментов для защиты данных будет способствовать укреплению доверия пользователей к современным цифровым технологиям и улучшению общей безопасности в сети.

# Публикация приложения

Приложение было загружено и опубликовано на GitHub [10].

# Список литературы

[1] Viacheslav - Краткое знакомство с Gradle [Электронный ресурс] (https://javarush.com/groups/posts/2126-kratkoe-znakomstvo-s-gradle). Дата обращения 11.11.2024

[2] pilot911 - Не бойтесь использовать Gradle [Электронный ресурс] (https://habr.com/ru/companies/itq\_group/articles/711712/). Дата обращения 11.11.2024

[3] Ben McCann - Using Gradle to build a JAR with dependencies [Электронный ресурс] (https://stackoverflow.com/questions/4871656/using-gradle-to-build-a-jar-with-dependencies). Дата обращения 11.11.2024

[4] mkyong - Log4j hello world example [Электронный ресурс] (https://mkyong.com/logging/log4j-hello-world-example/). Дата обращения 11.11.2024

[5] GFG - Java Swing | JTextField [Электронный ресурс] (https://www.geeksforgeeks.org/java-swing-jtextfield/). Дата обращения 11.11.2024

[6] The Java™ Tutorials - How to Use GridLayout [Электронный ресурс] (https://docs.oracle.com/javase/tutorial/uiswing/layout/grid.html). Дата обращения 11.11.2024

[7] Intellij - Build UI using Swing﻿ [Электронный ресурс] (https://www.jetbrains.com/help/idea/design-gui-using-swing.html). Дата обращения 11.11.2024

[8] Wikipedia - BASE64 [Электронный ресурс] (https://ru.wikipedia.org/wiki/Base64). Дата обращения 11.11.2024

[9] baeldung - SHA-256 [Электронный ресурс] (https://www.baeldung.com/sha-256-hashing-java). Дата обращения 11.11.2024

[10] mihailovily - Java-password-manager [Электронный ресурс] (https://github.com/mihailovily/Java-password-manager). Дата обращения 11.11.2024