Министерство науки и высшего образования РФ  
ФГАОУ ВПО

Национальный исследовательский технологический университет «МИСиС»

Институт Информационных технологий и компьютерных наук (ИТКН)

Кафедра Инфокоммуникационных технологий (ИКТ)

**Отчет по домашней работе**

по дисциплине «Технологии программирования»

на тему «Система хранения паролей и персональных данных»

Выполнил:  
студент группы ББИ-22-8

Абрамов Николай Иванович

Проверил:   
……………………..

Москва, 2023

**1. Формулирование требований**

**Общее описание**

Разработать приложение для безопасного и удобного хранения паролей и персональных данных пользователей. Это приложение будет обеспечивать защиту конфиденциальной информации, предоставляя возможность управления и организации данных в удобном интерфейсе.

**Перечень выполняемых функций**

1. Вход в систему:

* Логин с уникальным именем пользователя и паролем.
* Возможность изменения пароля.

2. Управление данными:

* Добавление, удаление и редактирование сохраненных паролей и личных

данных.

* Поиск по сохраненным записям.

3. Организация информации:

* Группировка паролей по категориям (например, работы, социальные

сети).

* Добавление ярлыков для быстрого поиска.

4. Показ информации:

* Просмотр списка записей с основной информацией.
* Подробный просмотр каждой записи для получения дополнительной

информации.

5. Сохранение и загрузка данных:

* Экспорт данных для создания копии безопасности.
* Импорт данных из ранее экспортированных файлов.

6. Настройки:

* Изменение параметров безопасности (например, длина паролей).
* Возможность изменения внешнего вида приложения.

**Требования к программе по каждой из функций, критерии качества выполнения**

1. Вход в систему:

Требования:

* Пользователи должны вводить логин и пароль для входа.
* Пароли должны быть защищены от посторонних.

Критерии качества:

* Доступ только после правильного ввода данных.
* Пароли должны храниться в безопасной форме.

2. Управление данными:

Требования:

* Добавление, удаление и редактирование записей.
* Данные не должны теряться при изменениях.

Критерии качества:

* Возможность без ошибок добавлять, удалять и изменять записи.
* Обновление данных без потерь.

3. Организация информации:

Требования:

* Группировка записей по категориям или тегам.
* Простой способ перемещаться между категориями.

Критерии качества:

* Данные должны быть легко упорядочены по группам.
* Удобство перехода между разделами.

4. Показ информации:

Требования:

* Доступ к полной информации по каждой записи.
* Краткий просмотр всех записей для быстрого обзора.

Критерии качества:

* Возможность получить всю информацию по запросу.
* Простой и информативный список записей.

5. Сохранение и загрузка данных:

Требования:

* Экспорт данных.
* Гарантия целостности информации при переносе.

Критерии качества:

* Успешный экспорт данных без потерь.
* Проверка на правильность передачи информации.

6. Настройки:

Требования:

* Изменение параметров безопасности и внешнего вида.

Критерии качества:

* Применение изменений без ошибок.
* Внешний вид должен измениться в соответствии с выбранными параметрами.

**Требования к интерфейсу**

1. Простота использования:

Требование: Понятный и легкий в освоении интерфейс.

Критерий качества: Пользователь может сразу понять, как добавить, изменить или удалить данные без лишних затруднений.

2. Ясное отображение информации:

Требование: Информация должна быть четко представлена.

Критерий качества: Пользователь видит все основные детали записей без необходимости искать дополнительную информацию.

3. Удобство для работы:

Требование: Простота работы с разными данными.

Критерий качества: Добавление, редактирование и удаление записей выполняется быстро и легко.

4. Хороший внешний вид:

Требование: Приятный внешний вид интерфейса.

Критерий качества: Дизайн приложения выглядит гармонично и не мешает пониманию информации.

5. Простая навигация:

Требование: Легкость перемещения по приложению.

Критерий качества: Пользователь может быстро переключаться между

разделами без путаницы.

**Входные, выходные данные и результаты работы программы**

Входные данные:

* Логин и пароль пользователя для доступа к системе.
* Новые записи с данными (логины, пароли, дополнительная информация)

для добавления в хранилище.

* Запросы на поиск, редактирование или удаление определенных записей.
* Настройки безопасности (например, предпочтительные символы в

пароле).

Выходные данные:

* Отображение списка всех записей с основной информацией (название,

категория, теги).

* Детальная информация о выбранной записи для просмотра или

редактирования.

* Экспорт данных.

Результаты работы программы:

* Успешный вход в систему при правильном вводе учетных данных.
* Добавление новых записей с паролями и персональными данными в

хранилище.

* Возможность изменения, удаления или поиска сохраненных данных.
* Организация и группировка данных для удобства использования.
* Экспорт данных для обеспечения безопасности

**Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание**

Рисунок 1 - Примерный интерфейс

**2. Анализ предметной области**

2.1 Хэширование паролей. Алгоритм SHA-256.

Что такое хеш по своей сути? Он является результатом использования хеш-функции на определенных входных данных и представляет собой строку символов фиксированной длины, которая кажется случайной. Хеш-функции спроектированы так, чтобы даже незначительные изменения в исходных данных приводили к радикальным изменениям в генерируемом хеше. Это делает хеш уникальным "отпечатком" для каждого набора данных.

SHA (Алгоритмы безопасного хеширования) – это семейство криптографических хэш-функций, способных принимать сообщения произвольной длины и вычислять уникальный хэш-код фиксированной длины. Хэш-код SHA может быть использован для проверки целостности сообщения, а также для генерации цифровой подписи сообщения. На данный момент существует несколько стандартов безопасного алгоритма, каждый последующий включает более надёжные хэш-функции.

SHA-256 относится к стандарту SHA-2, использует размер слова в 32 бита. Окончание 256 означает, что фиксированный размер хэша для любого сообщения равен 256 бит.

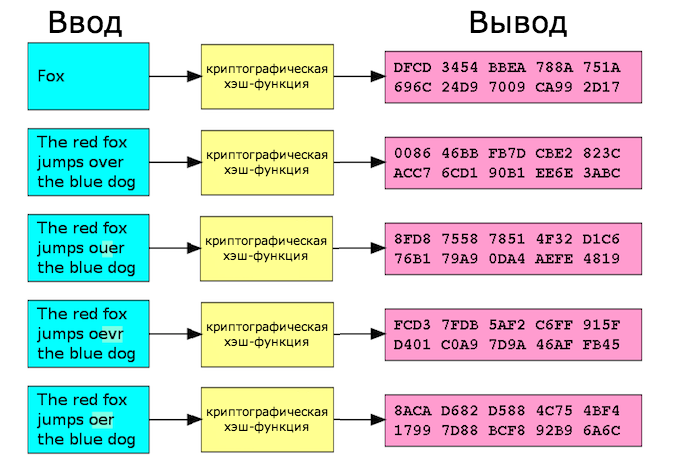


Рисунок 2 - Как работает хэширование

2.2 AES. шифрование данных.

AES (Advanced Encryption Standard) — это т. н. «симметричный» алгоритм шифрования, принятый в качестве стандарта и широко используемый в различных областях IT и вообще практически во всех сферах нашей, немыслимой без современных технологий, жизни.

Такую популярность AES завоевал благодаря (относительной) простоте, эффективности выполнения своей основной задачи по шифрованию информации, открытости, надёжности и проверенности временем и конечно благодаря стандартизации и признанию на официальном уровне.

В данный момент AES используется для шифрования информации в интернете, для передачи данных в различных сетях проводным и беспроводным способом, в различных устройствах и гаджетах и т. д., то есть везде, где нужно качественно шифровать информацию, затрачивая при этом минимум вычислительной мощности.

AES относится к так называемым «симметричным» алгоритмам, которые для шифрования и расшифровывания информации используют один и тот же криптографический ключ. Вообще, подобный тип шифрования является одним из самых распространённых и давно используемых.

При работе по принципу симметричного шифрования, секретный ключ имеется у обеих сторон взаимодействия (отправителя и получателя информации) и должен храниться и передаваться соответствующим безопасным образом.

Длина ключей в AES может выбираться из нескольких стандартных значений 128,192, 256 бит, в зависимости от решаемой задачи и необходимого уровня защиты данных.

Существуют также и другие алгоритмы шифрования, например, т. н. «несимметричное шифрование» в котором для шифрования и расшифровки используются разные ключи. Каждый тип алгоритма имеет свои достоинства и недостатки и свои области применения, в которых показывают себя наилучшим образом.

Шифр подстановки — это метод шифрования, в котором элементы исходного открытого текста заменяются зашифрованным текстом в соответствии с некоторым правилом. Элементами текста могут быть отдельные символы (самый распространённый случай), пары букв, тройки букв, комбинирование этих случаев и так далее.

Изображение выглядит как текст, снимок экрана, Шрифт, диаграмма

Автоматически созданное описание

Рисунок 3 - AES шифрование

Изображение выглядит как пингвин, мультфильм, текст, Нелетающая птица

Автоматически созданное описание

Рисунок 4 - AES (ECB простая замена) шифрование

2.3 Алгоритмы генерации паролей.

Устройство алгоритма: Мы будем брать адрес сайта и шифровать его с помощью алгоритма хеширования MD5. Результат шифрования и будет нашим паролем.

О хешировании мы уже писали, общий смысл такой: особый алгоритм шифрует текст таким образом, что по итоговому зашифрованному тексту невозможно восстановить исходный. Непонятна ни длина исходного текста, ни его содержимое.

Слабое место алгоритма только в том, что если зашифровать им один и тот же текст, получится один и тот же результат. То есть алгоритм работает предсказуемо: если зашифровать адрес mail.yandex.ru с помощью алгоритма MD5, всегда получится B81D1C770FD8F323B57CC73ED7B2546E. Это небезопасно.

Чтобы пароли были более уникальными, мы добавим к адресу сайта секретное слово, которое попросим у пользователя. И еще одно секретное слово впишем прямо в код программы. Вот эти два секретных слова и будут обеспечивать нашу безопасность пароля.

Всё, что осталось сделать — склеить адрес сайта и два секретных слова, зашифровать алгоритмом MD5, и вуаля — у нас будет уникальный трудночитаемый пароль, который никто никогда не подберет перебором.

Данный генератор был написан на языках HTML и CSS.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 5 - Примерный вид генератора паролей

Используемые источники:

* https://pro32.com/ru/article/chto-takoe-kheshirovanie-i-khesh-funktsii-tablitsy-paroli-i-fayly/#:~:text=%D0%A5%D0%B5%D1%88%20%E2%80%94%20%D1%8D%D1%82%D0%BE%20%D1%83%D0%BD%D0%B8%D0%BA%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F%20%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B0%20%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%BE%D0%B2,%22%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C%22%20%D0%B4%D0%BB%D1%8F%20%D1%8D%D1%82%D0%B8%D1%85%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85.
* https://habr.com/ru/articles/729260/
* https://ru.wikipedia.org/wiki/SHA-2
* https://electromicro.ru/resources/wiki/aes/aes1/
* https://habr.com/ru/articles/452042/
* https://www.ixbt.com/soft/alg-encryption-aes-2.shtml
* https://www.avast.ru/random-password-generator#pc
* https://thecode.media/qwerty123/
* https://ru.wikipedia.org/wiki/Bcrypt
* https://coderlessons.com/articles/php/pochemu-vy-dolzhny-ispolzovat-
* bcrypt-dlia-kheshirovaniia-sokhranennykh-parolei

**3. Разработка алгоритмов решения задачи**

3.1 Хэширование паролей

Хэширование паролей — это процесс преобразования пароля в некоторую строку фиксированной длины (хэш), которая не может быть обратно преобразована в исходный пароль. Это защищает пароли от несанкционированного доступа, поскольку даже при доступе к хэшу сложно восстановить исходный пароль.

Пример хэширования пароля с использованием библиотеки BCrypt.Net:

using BCrypt; // Импортируем библиотеку для работы с хэшированием паролей

class PasswordManager

{

// Метод для хэширования пароля

public string HashPassword(string password)

{

// Создаем хэш пароля с помощью BCrypt

string hashedPassword = BCrypt.Net.BCrypt.HashPassword(password);

return hashedPassword; // Возвращаем хэшированный пароль

}

// Метод для проверки правильности пароля

public bool VerifyPassword(string inputPassword, string hashedPassword)

{

// Проверяем, соответствует ли введенный пароль его хэшу

bool isPasswordValid = BCrypt.Net.BCrypt.Verify(inputPassword, hashedPassword);

return isPasswordValid; // Возвращаем результат проверки

}

}

// Пример использования класса PasswordManager для хэширования и проверки пароля

PasswordManager passwordManager = new PasswordManager();

string userPassword = "user\_password";

// Хэширование пароля

string hashedPassword = passwordManager.HashPassword(userPassword);

// Проверка правильности пароля

bool isPasswordCorrect = passwordManager.VerifyPassword(userPassword, hashedPassword);

* Для использования этой библиотеки нужно установить пакет BCrypt.Net

через NuGet Package Manager.

* HashPassword: Этот метод библиотеки BCrypt.Net принимает пароль в

качестве аргумента и возвращает его хэш.

* VerifyPassword: Этот метод сравнивает введенный пользователем

пароль с хэшем, возвращая true, если они совпадают.

Алгоритм хэширования BCrypt, как правило, имеет временную сложность, зависящую от параметра (work factor), который определяет количество итераций, выполняемых алгоритмом. Увеличение значения work factor увеличивает количество итераций хэширования, что делает процесс более медленным, но повышает безопасность пароля.

Обычно сложность алгоритма BCrypt регулируется таким образом, чтобы время хэширования оставалось приемлемым для обычного использования (в рамках нескольких сотен миллисекунд), но при этом усложняло процесс взлома паролей.

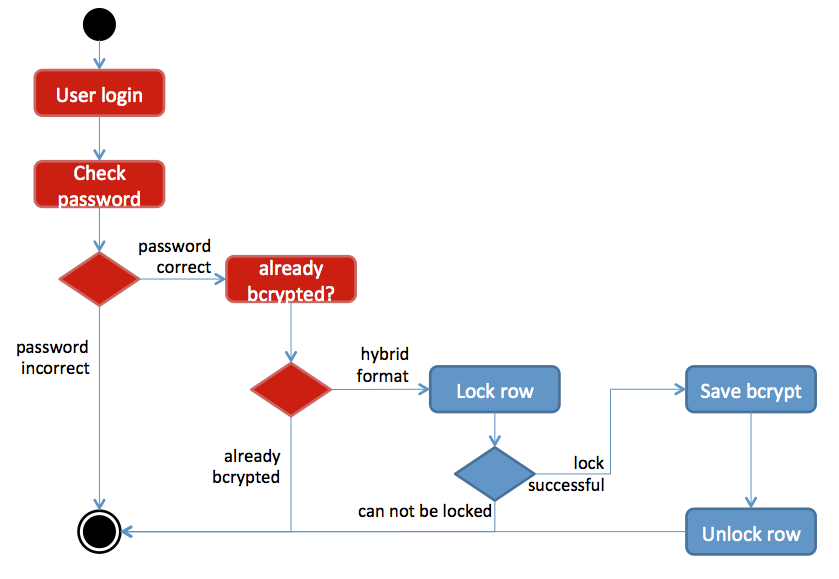


Рисунок 6 - Блок-схема для bcrypt

3.2 Шифрование данных.

Шифрование данных – это процесс преобразования информации в непонятную форму с использованием специальных алгоритмов (шифров), чтобы только авторизованные пользователи могли прочитать эти данные с помощью специального ключа.

Вот более подробный пример работы с шифрованием данных на примере алгоритма AES (Advanced Encryption Standard):

using System;

using System.Security.Cryptography;

using System.IO;

class DataEncryption

{

// Метод для шифрования данных

public byte[] EncryptData(string data, byte[] key, byte[] iv)

{

byte[] encryptedData;

using (Aes aesAlg = Aes.Create()) // Создание экземпляра AES

{

aesAlg.Key = key; // Установка ключа

aesAlg.IV = iv; // Установка вектора инициализации (IV)

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV); // Создание объекта для шифрования

using (MemoryStream msEncrypt = new MemoryStream()) // Создание потока для шифрования

{

using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write)) // Подключение потока шифрования к потоку данных

{

using (StreamWriter swEncrypt = new StreamWriter(csEncrypt)) // Создание писателя для записи зашифрованных данных

{

swEncrypt.Write(data); // Запись данных в поток шифрования

}

encryptedData = msEncrypt.ToArray(); // Получение зашифрованных данных из памяти

}

}

}

return encryptedData; // Возвращение зашифрованных данных

}

}

“EncryptData”, который принимает строку данных, ключ и вектор инициализации (IV) для шифрования. Он использует алгоритм AES для создания шифратора, который шифрует данные и возвращает зашифрованный массив байтов.

Дешифрование:

using System.Security.Cryptography;

class DataDecryption

{

// Метод для дешифрования данных

public string DecryptData(byte[] encryptedData, byte[] key, byte[] iv)

{

string decryptedData = null;

using (Aes aesAlg = Aes.Create()) // Создание экземпляра AES

{

aesAlg.Key = key; // Установка ключа

aesAlg.IV = iv; // Установка вектора инициализации (IV)

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV); // Создание объекта для дешифрования

using (MemoryStream msDecrypt = new MemoryStream(encryptedData)) // Создание потока для дешифрования

{

using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read)) // Подключение потока дешифрования к потоку зашифрованных данных

{

using (StreamReader srDecrypt = new StreamReader(csDecrypt)) // Создание читателя для чтения зашифрованных данных

{

decryptedData = srDecrypt.ReadToEnd(); // Чтение данных из потока дешифрования

}

}

}

}

return decryptedData; // Возвращение расшифрованных данных

}

}

DecryptData, который принимает зашифрованный массив байтов, ключ и вектор инициализации для дешифрования данных с использованием алгоритма AES.

Алгоритм AES является симметричным алгоритмом шифрования, и его временная сложность, как правило, зависит от выбранного режима и размера ключа. Основной алгоритм AES обычно имеет очень высокую производительность на современных компьютерах и устройствах. Время выполнения одной операции шифрования или дешифрования обычно очень мало и варьируется в пределах наносекунд или микросекунд в зависимости от конкретной реализации алгоритма и характеристик системы.

Изображение выглядит как текст, диаграмма, План, линия

Автоматически созданное описание

Рисунок 7 - Блок-схема AES алгоритма

3.3 Генерация безопасных паролей.

Пример кода, который генерирует безопасный пароль:

using System;

using System.Security.Cryptography;

class PasswordGenerator

{

// Метод для генерации безопасного пароля указанной длины

public string GenerateRandomPassword(int length)

{

// Набор символов, которые могут быть использованы в пароле

const string validChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&\*()\_+-=[]{};:,.<>?";

char[] chars = new char[length]; // Создаем массив для хранения символов пароля

byte[] randomData = new byte[length]; // Создаем массив байт для генерации случайных данных

// Используем криптографически-безопасный генератор случайных чисел для заполнения массива randomData

using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())

{

rng.GetBytes(randomData);

}

// Создаем пароль, выбирая случайные символы из validChars в соответствии с данными из randomData

for (int i = 0; i < length; i++)

{

chars[i] = validChars[randomData[i] % validChars.Length];

}

return new string(chars); // Возвращаем сгенерированный пароль

}

}

// Пример использования класса PasswordGenerator для генерации пароля

PasswordGenerator passwordGenerator = new PasswordGenerator();

string generatedPassword = passwordGenerator.GenerateRandomPassword(12); // Генерация пароля длиной 12 символов

Console.WriteLine("Сгенерированный пароль: " + generatedPassword);

Класс PasswordGenerator, который содержит метод GenerateRandomPassword, созданный для генерации безопасных паролей заданной длины. Он использует криптографически-безопасный генератор случайных чисел для создания пароля из набора символов, включающего буквы в разных регистрах, цифры и некоторые специальные символы.

Временная сложность алгоритма генерации пароля примерно линейна и зависит от его длины. Чем длиннее пароль, тем больше времени требуется на его создание. В общем случае, для генерации пароля длиной n символов временная сложность может быть приблизительно O(n).

**4. Проектирование диаграмм**

4.1 Запуск приложения:

* Начальное состояние приложения. Ожидает ввода данных от

пользователя.

4.2 Ожидание ввода логина или пароля:

* Пользователю предлагается ввести логин и пароль для доступа к базе

данных.

* События:

Пользователь вводит логин и пароль.

Пользователь нажимает кнопку "Войти в базу данных" или "Создать безопасный пароль".

4.3 Вход в базу данных:

* После ввода логина и пароля.
* Событие: Успешный вход в базу данных.

4.4 Просмотр базы данных:

* Пользователь видит таблицу с полями ID, Login, Password, Website.
* Событие: Пользователь выбирает одно из следующих действий:

1. Нажимает кнопку "Сохранить" для сохранения изменений.
2. Нажимает кнопку "Добавить новую строку" для создания новой записи.
3. Нажимает кнопку "Экспортировать" для экспорта данных.

4.5 Завершение работы:

* Окончание сеанса работы с базой данных.
* Пользователь выходит из приложения или завершает сеанс работы.

Эта диаграмма отображает основные состояния и переходы между ними в вашем приложении для управления базой данных паролей. Каждый переход обусловлен определенными действиями пользователя или событиями, которые происходят во время использования приложения.

Изображение выглядит как диаграмма, текст, линия, План

Автоматически созданное описание

Рисунок 8 - Диаграмма для интерфейса

**5. Кодирование модулей**

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 9 - Form1 начальная страница

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 10 - Form 1 генерация надежного пароля

Изображение выглядит как текст, снимок экрана, программное обеспечение, веб-страница

Автоматически созданное описание

Рисунок 11 - Form 1 хэширование пароля

Изображение выглядит как текст, снимок экрана, дисплей, число

Автоматически созданное описание

Рисунок 12 - Form 2 главная страница

Изображение выглядит как текст, программное обеспечение, число, Шрифт

Автоматически созданное описание

Рисунок 13 - Нажатие кнопки экспорт, с последующим открытием файла в excel таблице

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 14 - Нажатие кнопки "сохранить"

Программный код для Form 1

Этот код представляет собой пример простой программы для учета и хранения паролей. Здесь реализована форма Form1, в которой пользователь может ввести логин и пароль.

* GenerateSecurePassword(): Метод для генерации надежного пароля. В

текущем примере создается случайная строка из различных символов.

* HashPassword(string password): Метод для хэширования введенного

пароля с использованием алгоритма SHA256.

* loginButton\_Click\_1(object sender, EventArgs e): Обработчик события

нажатия кнопки "Войти". В этом обработчике введенный пароль хэшируется и выводится в виде строки сообщения для демонстрации.

* createPasswordButton\_Click\_1(object sender, EventArgs e): Обработчик

события нажатия кнопки "Создать пароль". Генерирует новый надежный пароль и автоматически вставляет его в текстовое поле для пароля.

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Учет\_и\_хранение\_паролей

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private string GenerateSecurePassword()

{

// Здесь можно реализовать логику для генерации надежного пароля

// В данном примере генерируется случайная строка длиной 12 символов

const string validChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890!@#$%^&\*()\_-+=[]{}<>"; // Допустимые символы для пароля

StringBuilder password = new StringBuilder();

using (RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider())

{

byte[] uintBuffer = new byte[sizeof(uint)];

int length = 12;

while (length-- > 0)

{

rng.GetBytes(uintBuffer);

uint num = BitConverter.ToUInt32(uintBuffer, 0);

password.Append(validChars[(int)(num % (uint)validChars.Length)]);

}

}

return password.ToString();

}

private string HashPassword(string password)

{

using (SHA256 sha256Hash = SHA256.Create())

{

byte[] bytes = sha256Hash.ComputeHash(Encoding.UTF8.GetBytes(password));

StringBuilder builder = new StringBuilder();

for (int i = 0; i < bytes.Length; i++)

{

builder.Append(bytes[i].ToString("x2"));

}

return builder.ToString();

}

}

private void loginButton\_Click\_1(object sender, EventArgs e)

{

string enteredLogin = loginTextBox.Text;

string enteredPassword = passwordTextBox.Text;

// Предположим, проверка всегда успешна для примера

bool loginSuccessful = true;

string hashedPassword = HashPassword(enteredPassword);

MessageBox.Show("Хэшированный пароль: " + hashedPassword);

if (loginSuccessful)

{

// Переход на вторую форму после успешного входа

Form2 databaseForm = new Form2();

databaseForm.Show();

this.Hide(); // Скрываем текущую форму

}

else

{

MessageBox.Show("Неверный логин или пароль. Попробуйте снова.");

loginTextBox.Text = "";

passwordTextBox.Text = "";

}

}

private void createPasswordButton\_Click\_1(object sender, EventArgs e)

{

string generatedPassword = GenerateSecurePassword();

passwordTextBox.Text = generatedPassword; // Вставка сгенерированного пароля в поле пароля

MessageBox.Show("Создан новый надежный пароль и автоматически вставлен в поле.");

}

}

}

Этот код представляет собой форму Form2, которая служит для отображения базы данных паролей в виде таблицы и обеспечивает основные функции управления этой базой данных.

В конструкторе формы (public Form2()):

* Создается объект DataTable (databaseTable), который представляет собой

таблицу с четырьмя колонками: "Код", "Логин", "Пароль" и "Сайт".

* В этой таблице добавляются строки с примерами данных.
* Таблица устанавливается в качестве источника данных для

DataGridView (таблицы на форме).

AddNewRowButton\_Click:

* Обработчик события нажатия кнопки "Добавить новую запись".
* Создает новую запись в базе данных и добавляет ее в таблицу.

ExportButton\_Click\_1:

* Обработчик события нажатия кнопки "Экспортировать".
* Создает CSV-файл с данными из таблицы, который пользователь может

сохранить на компьютере.

SaveButton\_Click\_1:

* Обработчик события нажатия кнопки "Сохранить". В данном примере

выводит сообщение о сохранении изменений (без реального сохранения внесенных данных в базу).

Программный код для Form 2

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 Учет\_и\_хранение\_паролей

{

public partial class Form2 : Form

{

private DataTable databaseTable;

public Form2()

{

InitializeComponent();

//databaseTable = data;

// Привязка таблицы к DataGridView на форме

DataTable databaseTable = new DataTable();

databaseTable.Columns.Add("Код", typeof(int));

databaseTable.Columns.Add("Логин", typeof(string));

databaseTable.Columns.Add("Пароль", typeof(string));

databaseTable.Columns.Add("Сайт", typeof(string));

// Добавляем строки с данными

databaseTable.Rows.Add(1, "Maksik", "dffafac1", "Яндекс");

databaseTable.Rows.Add(2, "Sashik", "qwert", "Фэйсбук");

databaseTable.Rows.Add(3, "Ignat", "HNFAS111", "Gmail");

databaseTable.Rows.Add(4, "pokest14", "hiUOU134!!!", "VK");

databaseTable.Rows.Add(5, "hazbik", "156w1d41413", "Одноклассники");

databaseTable.Rows.Add(6, "vladislav", "et,ezgfjrhg", "Ватсапп");

// Устанавливаем эту таблицу в качестве источника данных для DataGridView

dataGridView.DataSource = databaseTable;

}

private void Form2\_Load(object sender, EventArgs e)

{

}

private void dataGridView\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void AddNewRowButton\_Click(object sender, EventArgs e)

{

// Добавление новой записи в базу данных

DataRow newRow = databaseTable.NewRow();

newRow["ID"] = databaseTable.Rows.Count + 1;

newRow["Login"] = "NewUser";

newRow["Password"] = "NewPassword";

newRow["Website"] = "NewWebsite.com";

databaseTable.Rows.Add(newRow);

MessageBox.Show("Новая запись добавлена.");

}

private void ExportButton\_Click\_1(object sender, EventArgs e)

{

SaveFileDialog saveDialog = new SaveFileDialog();

saveDialog.Filter = "CSV файл (\*.csv)|\*.csv";

saveDialog.Title = "Сохранить как CSV";

if (saveDialog.ShowDialog() == DialogResult.OK)

{

DataTable table = (DataTable)dataGridView.DataSource;

string delimiter = ",";

// Создание строки заголовков

string columnHeaderText = "";

int countColumn = table.Columns.Count;

for (int i = 0; i < countColumn; i++)

{

columnHeaderText += table.Columns[i].ColumnName + delimiter;

}

columnHeaderText = columnHeaderText.Remove(columnHeaderText.Length - 1);

// Запись заголовков в файл

System.IO.File.WriteAllText(saveDialog.FileName, columnHeaderText + Environment.NewLine);

// Запись данных строк в файл

foreach (DataRow row in table.Rows)

{

string rowData = "";

for (int i = 0; i < countColumn; i++)

{

rowData += row[i].ToString() + delimiter;

}

rowData = rowData.Remove(rowData.Length - 1);

System.IO.File.AppendAllText(saveDialog.FileName, rowData + Environment.NewLine);

}

MessageBox.Show("Данные успешно экспортированы в CSV файл.");

}

}

private void SaveButton\_Click\_1(object sender, EventArgs e)

{

// Сохранение изменений в базе данных

MessageBox.Show("Изменения сохранены.");

}

}

}

**6. Структурное тестирование**

Описание методики.

Цель методики: Убедиться, что программа работает так, как нужно.

Что включает:

* Подготовка: Какие тесты проводить, когда и с какими данными.
* Тесты: Шаги для проверки программы.
* Отчеты: Что нашли и как оценить результаты.

Какие части:

* Что тестировать: Какие части программы проверять.
* Как проверять: Какие тесты проводить, какие данные использовать.
* Что делать с результатами: Как записать найденные ошибки и как их

исправить.

Инструменты:

* Тестовые сценарии: Описание шагов тестирования.
* Программы для тестирования: Инструменты, которые помогают

проверять программу.

* Отчеты: Документы о том, что проверили и что нашли.

6.1 Потоковой граф для loginButton\_Click:

* Начало: Нажатие кнопки "Войти".
* Ввод данных: Ввод логина и пароля.
* Проверка данных: Проверка логина и пароля.

Если данные верные: Переход на Form2 (вторую форму).

Если данные неверные: Отображение сообщения об ошибке.

6.2 Потоковой граф для SaveButton\_Click:

* Начало: Нажатие кнопки "Сохранить".
* Проверка данных: Проверка корректности введенных данных.
* Сохранение: Сохранение изменений в базе данных.
* Отчет: Отображение сообщения о успешном сохранении.

6.3 Потоковой граф для AddNewRowButton\_Click:

* Начало: Нажатие кнопки "Добавить новую строку".
* Создание новой строки: Создание пустой строки в таблице.
* Заполнение данных: Ввод данных в новую строку.
* Добавление в таблицу: Добавление новой строки в таблицу.
* Отчет о добавлении: Отображение сообщения о успешном добавлении

новой записи.

6.4 Потоковой граф для ExportButton\_Click:

* Начало: Нажатие кнопки "Экспортировать".
* Выбор файла для экспорта: Открытие окна выбора места сохранения.
* Формирование файла: Запись заголовков и данных таблицы в файл.
* Сохранение файла: Сохранение файла в выбранном месте.
* Отчет об экспорте: Отображение сообщения об успешном экспорте

данных в файл.

6.5 Потоковой граф для GenerateSecurePassword():

* Начало: Запуск метода GenerateSecurePassword().
* Генерация символов: Создание допустимого набора символов для

пароля.

* Генерация случайной строки: Получение случайной строки длиной 12

символов.

* Получение случайных чисел: Генерация случайных чисел для

формирования пароля.

* Преобразование чисел в символы: Преобразование чисел в символы,

используя допустимый набор.

* Формирование пароля: Создание надежного пароля.

6.6 Потоковой граф для HashPassword():

* Начало: Запуск метода HashPassword() с входным паролем.
* Кодирование в UTF-8: Преобразование пароля в последовательность

байтов по схеме кодировки UTF-8.

* Хеширование: Применение алгоритма SHA-256 к байтам пароля.
* Преобразование в строку: Преобразование полученного хеша в строку.

Для метода GenerateSecurePassword():

* Входные данные: нет прямых параметров в методе, но предположим,

что он должен генерировать надежные пароли.

* Условия запуска: вызов метода GenerateSecurePassword().
* Ожидаемые результаты: Возврат строки сгенерированного надежного

пароля. Ожидается, что пароль будет длиной 12 символов и будет содержать

символы из различных категорий (буквы верхнего и нижнего регистра, цифры,

специальные символы).

Для метода HashPassword():

* Входные данные: строка пароля.
* Условия запуска: вызов метода HashPassword() с передачей строки

пароля.

* Ожидаемые результаты: Возврат строки, содержащей хэш-значение

переданного пароля. Ожидается, что хэш-значение будет представлено в

формате SHA256 и будет уникальным для каждого уникального пароля.

**7. Функциональное тестирование**

Методика тестирования GenerateSecurePassword():

1. Тестирование на длину пароля:

* Описание: Проверка, что генерируемый пароль имеет заданную длину.
* Шаги: Вызов метода GenerateSecurePassword() без параметров.
* Ожидаемый результат: Получение строки, длина которой равна 12

символам.

2. Тестирование разнообразия символов:

* Описание: Проверка, что пароль содержит символы различных

категорий.

* Шаги: Многократные вызовы GenerateSecurePassword().
* Ожидаемый результат: Получение паролей, содержащих буквы

верхнего и нижнего регистра, цифры, специальные символы.

3. Тестирование использования определенных символов:

* Описание: Проверка, что метод использует определенные символы при

генерации пароля.

* Шаги: Передача специальных символов, цифр, букв в метод

GenerateSecurePassword().

* Ожидаемый результат: Генерация паролей, содержащих переданные

символы.

Методика тестирования HashPassword():

1. Тестирование хэширования:

* Описание: Проверка корректности хэширования различных паролей.
* Шаги: Передача различных паролей в метод HashPassword().
* Ожидаемый результат: Получение уникального хэша для каждого

уникального пароля.

2. Тестирование безопасности:

* Описание: Проверка обработки некорректных входных данных.
* Шаги: Передача пустой строки, короткого пароля, специальных

символов.

* Ожидаемый результат: Корректное хэширование для правильных

данных и обработка некорректных как ошибочных.

3. Тестирование производительности:

* Описание: Проверка производительности метода при большом объеме

данных.

* Шаги: Многократные вызовы HashPassword() для большого количества

данных.

* Ожидаемый результат: Работоспособность метода без существенного

ухудшения производительности.

**Заключение**

В ходе работы были рассмотрены ключевые аспекты:

* Интерфейс и функционал: Разработаны две формы - одна для ввода

логина и пароля, другая для отображения и управления базой данных. Взаимодействие позволяет пользователям безопасно хранить данные.

* Шифрование и хэширование: Реализованы методы генерации

безопасных паролей и хэширования паролей для обеспечения дополнительного уровня безопасности.

* Тестирование: Разработаны методики тестирования для проверки

корректности работы генератора паролей и процесса хэширования.

* Экспорт данных: Предусмотрена возможность экспорта данных для

сохранения или передачи безопасным способом.

* Алгоритмы и безопасность: Использованы алгоритмы хэширования и

шифрования для защиты паролей и конфиденциальных данных.