РАЗРАБОТКА ПРИНЦИПОВ ПОСТРОЕНИЯ СИСТЕМ ХРАНЕНИЯ КОНФИДЕНЦИАЛЬНОЙ ИНФОРМАЦИИ ДЛЯ ЛИЧНОГО И КОРПОРАТИВНОГО ИСПОЛЬЗВАНИЯ

Некрасов Е.П., студент гр. ИТм-181, II курс
Научный руководитель: Ванеев О.Н., к.т.н., доцент
Кузбасский государственный технический университет
имени Т.Ф. Горбачева
г. Кемерово

В современном мире социальных сетей, разнообразных тематических форумов и ресурсов, вопрос о том, как безопасно хранить пароли, является достаточно актуальным. Для того, чтобы создать максимально безопасный пароль для каждого важного ресурса, потребуется немного времени и фантазии, однако запомнить такое количество безопасных паролей человеку будет достаточно сложно, если только он не обладает супер способностями своей памяти. Выхода может быть два - или не заморачиваться с безопасностью, используя по старинке один пароль для всех сайтов, состоявший из названия своего имени и дня рождения, или прибегнуть к помощи специального программного обеспечения.

Менеджер паролей — это программа, которая шифрует и хранит ваши пароли в надёжном месте. Два в одном: безопасные пароли, которыми удобно пользоваться. Остаётся запомнить единственный пароль — к самому менеджеру. Проведя детальный анализ существующих прототипов менеджеров паролей было выявлено следующее:

- Сохранение конфиденциальной информации — это важная и актуальная проблема;
- Большинство менеджеров паролей направлены исключительно хранению паролей, однако остальная конфиденциальная информация не затрагивается;
- Очень мало менеджеров паролей, которые использовали бы принцип 2fa двухфакторная аутентификация, которая стала своего рода стандартом в мире защиты данных;
- Локальные менеджеры паролей не подходят для корпоративного использования, так как представляют собой один файл, пользоваться которым может единственный человек, или файл на сервере, который чаще всего совсем не защищен от хищения
- Облачные же требуют дополнительного программного обеспечения
- Некоторые из менеджеров паролей имеют старый и уж очень недружелюбный интерфейс;

На основе данных недостатков можно сформулировать задачу - разработать систему, которая хранит конфиденциальную информацию в надежном формате и имеет ряд особенностей:

- Позволяет хранить в себе не только пароли, но и различные текст, мелкие файлы;
- Не требует дополнительного ПО;
- Бесплатный;
- Подходит для корпоративного использования;
- Наличие двухфакторной авторизации;
- Простой и понятный интерфейс;
- Открытый исходный код;
- Синхронизация с большим количеством устройств;

Из данных особенностей можно сформировать диаграмму требований к системе. Она показана на рисунке 1

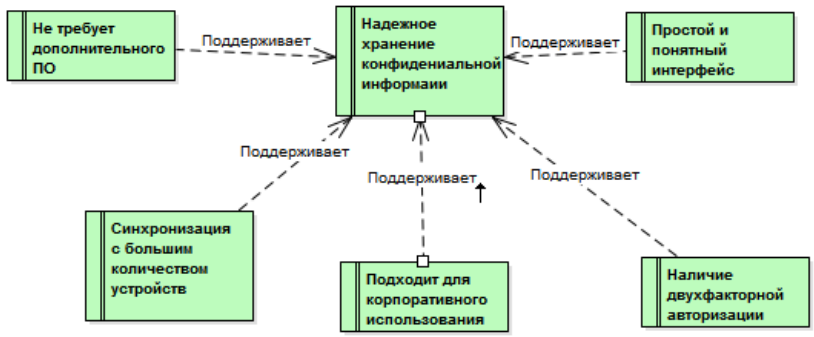

Рисунок 1 – Диаграмма вариантов использования к системе

Перед переходом к реализаии требуется провести анализ технологий, в первую очередь определиться с алгоритмом шифрования от которого будет зависеть эффективность системы.

Следующий этап — это подбор алгоритма шифрования, в .NET Framework классы реализуют несколько симметричных и асимметричных алгоритмов. Симметричные алгоритмы работают быстро в обоих направлениях, однако они имеют ряд проблем, связанные с ними:

- Необходим безопасный способ передачи ключа.
- Периодически нужно менять ключи.
- Ключ должен храниться в безопасном месте.

Асимметричные алгоритмы пытаются решить проблемы, присущие симметричным алгоритмам. Они основаны на математических методах, которые требуют разных ключей для шифрования и расшифровки. RSA поддерживает прямое шифрование и расшифровку значений. Большой проблемой асимметричных алгоритмов является то, что они работают намного медленнее (в зависимости от шифруемых данных) симметричных. Технологии, подобные SSL, используют асимметричные алгоритмы в начале, при установлении сеанса связи. На начальных шагах обмена трафик между клиентом и сер-

вером защищается асимметричным шифрованием. На данном этапе клиент и сервер могут безопасно обменяться симметричным ключом. Это позволяет объединить преимущества симметричного и асимметричного шифрования.

Для достижения максимальной «защиты» данных пользователя следует обратиться к системе двухфакторной аутентификации — это метод идентификации пользователя в каком-либо сервисе (как правило, в Интернете) при помощи запроса аутентификационных данных двух разных типов, что обеспечивает двухслойную, а значит, более эффективную защиту аккаунта от несанкционированного проникновения. Впрочем, двухфакторная защита не означает что аккаунт невозможно будет украсть, однако она значительно усложняет процесс кражи данных, так как даже если пароль был скомпрометирован, злоумышленнику придется или раздобыть мобильник жертвы.

В рамках данной системы для реализации будет использоваться библиотека .Net - TwoFactorAuth.Net, которая доступна в виде пакета NuGet. Используется для двухфакторной (или многофакторной) аутентификации с использованием TOTP и QR-кодов

Определившись с механизмом сохранения и хранения данных можно приступить к выбору платформы которая способна удовлетворить все требования. Так как передо мной стоит задача разработки приложения, которое в первую очередь направлено на сохранность персональных данных пользователей, было принято решение свести к минимуму централизованное хранение данных. Наиболее привлекательный вариант — это хранение конфиденциальной информации непосредственно на компьютере пользователей, однако в максимально недоступном виде. Исходя из всего вышесказанного необходима и достаточна разработка классического настольного приложения. На данный момент Windows представляет три основные платформы приложений каждая из которых имеет свои преимущества и недостатки.

- Универсальная платформа Windows (UWP)
- WPF (.NET);
- Windows Forms (.NET);

Windows Forms — это исходная платформа для управляемых приложений для Windows на основе упрощенной модели пользовательского интерфейса с доступом ко всем компонентам платформы .NET Core или полной платформы .NET Framework. Однако любое современное приложение обязано иметь приветливый и интуитивно понятный интерфейс чего обеспечить Windows Forms не может.

UWP — это передовая платформа для приложений и игр, предназначенных для Windows 10. Это платформа с широкими возможностями настройки, которая использует разметку XAML для отделения пользовательского интерфейса (представления) от кода (бизнес-логики). Хотя UWP и является на данный момент новейшей, платформой приложений распространенность Windows 10 в соотношении с другими платформами оставляет желать лучшего, а разрабатываемое приложение направлено на широкую аудиторию пользователей.

WPF — это общепризнанная платформа для управляемых приложений для Windows с доступом ко всем компонентам платформы .NET Core или полной платформы .NET Framework. Она также использует разметку XAML для отделения пользовательского интерфейса от кода. Эта платформа создана для классических приложений, для которых требуется расширенный пользовательский интерфейс, настройка стилей и сценарии с большим объемом графики. Плюсом также является и то что переход с WPF на UWP достаточно безболезнен, так что в случае необходимости есть возможность перенаправить приложение под систему Windows 10.

Сам принцип разработки WPF приложений особо ничем не отличается от обычных Windows Forms, мы просто получаем больше возможностей по созданию форм благодаря XAML. Данные попадают на форму благодаря привязкам, то есть перед тем как попасть на форму, в одном классе, а то и в одном методе проходит получение, подготовка и распределение данных по форме, что существенно осложняет не только доработку приложения, но и страдает быстродействие всей системы.

Поэтому было принято решение разработки основываясь на не так давно появившемся паттерне MVVM, который разделяет приложение на три компоненты: модель, представление, бизнес-логика(модель-представление). Прослеживается некоторая аналогия паттерну MVC однако в данном случае ни модель, ни представление ни сам компонент бизнес-логики не знают друг о друге. Грубо говоря View знает только то что во ViewModel может быть некое тектовое свойство которое её следует брать для элемента формы. Другими словами, наш интерфейс не как не зависит от кода, что позволяет к примеру, разрабатывать двумя разными командами интерфейс и код по отдельности, либо изменить интерфейс убрав из него не нужные элементы (не трогая при этом код). В этом и суть разделения всего по слоям.

Перед тем как рассматривать на конкретном примере, нужно провести подготовку приложения. Следует внести отдельный класс который будет отвечать за связь между View и ViewModel, например BaseModelView. Он представлен на рисунке 2

```
public class BaseModelView : INotifyPropertyChanged
{

    public event PropertyChangedEventHandler PropertyChanged;

    protected bool Set<T>(ref T field, T value, [CallerMemberName] string propertyName = null)
    {
        if (EqualityComparer<T>.Default.Equals(field, value))
            return false;

        field = value;
        NotifyPropertyChanged(propertyName);
        return true;
    }

    protected void NotifyPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

Рисунок 2 – Листинг метода связки свойств между View и ViewModel

В данном классе следует обратить внимание на интерфейс INotifyPropertyChanged, введенный в .NET Framework начиная с версии 2 и выше. Этот интерфейс реализует систему уведомлений, которая активируется, когда значение свойства изменяется. Это требуется в модели-представления, чтобы сделать механизм привязки пользовательского интерфейса XAML динамическим. Другими словами, View узнает о свойстве ViewModel в тот момент, когда получает уведомление об изменении этого свойства. Так, как наша View должна быть независима от ViewModel мы не можем просто так использовать Click и другие события. Вот на помощь к нам приходят команды, мы создаем свойство данной команды и привязываемся и даже если его нет - наша программа не упадет и будет работать как надо. Второй класс – RelayCommand показан на рисунке 3

```
public class RelayCommand<T> : ICommand
{
    private Action<T> action;
    public RelayCommand(Action<T> action) => this.action = action;
    public bool CanExecute(object parameter) => true;
    #pragma warning disable CS0067
    public event EventHandler CanExecuteChanged;
    #pragma warning restore CS0067
    public void Execute(object parameter) => action((T)parameter);
}
```

Рисунок 3 – Листинг метода связки команд между View и ViewModel

Далее можно рассмотреть реализацию на примере зашифрованных файлов. Основной класс с бизнес-логикой основан на ранее созданном Base-ModelView. Пример его реализации показана на рисунке 4

```
}

public class ConfidentialInformationView : BaseModelView
{
    public enum Statuses
    {
        Shared,
        Private,
        Deleted
    }

    public ConfidentialInformationView(string id, string filePath,string title, DateTime dateUpdate, string parent, Statuses status, string key)
    {
        this.id = id;
        this.title = title;
        this.dateUpdate = dateUpdate;
        this.parent = parent;
        this.status = status;
        this.key = key;
        this.filePath = filePath;
        RenameCommand = new RelayCommand<string>(Rename);
        DeleteCommand = new RelayCommand(Delete);
        ShareCommand = new RelayCommand<string[]>(Share);
        ReplaceCommand = new RelayCommand<string>(Replace);
        EncryptCommand = new RelayCommand<string[]>(Encrypt);
    }
```

Рисунок 4 – Листинг примера реализации класса логики в MVVM

Каждое свойство обладает своим геттером и сеттером. Это показано на рисунке 5

```
private string id;
public string Id
{
    get => id;
    set => Set(ref id, value);
}

private string filePath;
public string FilePath
```

Рисунок 5 – Листинг примера указания свойств класса

Методы задаются с типом ICommand также с разной доступностью. Это показано на рисунке 6

```
public ICommand ShareCommand { get; set; }
private void Share(string[] InputParameters)
{
    string fileName = InputParameters[0];
    string SharedUserName = InputParameters[1];
    status = Statuses.Deleted;
    // ОТПРАВИТЬ ФАЙЛ НА СЕРВЕР, ГДЕ ОН ПРОЛЕЖИТ 2 ДНЯ
    HttpWebRequest request = (HttpWebRequest)WebRequest.Create("http://127.0.0.1/post/getfile?usrName=" + SharedUserName + ".php");
    request.ContentType = "multipart/form-data;";
    request.Method = "POST";
    request.KeepAlive = true;
    request.Credentials = CredentialCache.DefaultCredentials;
    Stream requestStream = request.GetRequestStream();

    string headerTemplate = "Content-Disposition: form-data; name=\"{0}\"; filename=\"{1}\"\r\nContent-Type: {2}\r\n\r\n";
    string header = string.Format(headerTemplate, "file", fileName, "text/txt");
    byte[] headerbytes = Encoding.UTF8.GetBytes(header);
    requestStream.Write(headerbytes, 0, headerbytes.Length);
    string decryptionFileName = fileName.Replace("encrypt_", "");

    CryptoHelper.DecryptFile(fileName, decryptionFileName);
    FileStream fileStream = new FileStream(decryptionFileName, FileMode.Open, FileAccess.Read);
```

Рисунок 6 – Листинг примера команды во ViewModel

В XAML доступ осуществляется через привязку Binding. Это показано на рисунке 7

```
<Border
                BorderBrush="#e6e8eb"
                BorderThickness="0 0 0 1"
                Grid.Column="2"
                HorizontalAlignment="Center"
                VerticalAlignment="Center">
    <Label Foreground="#1b2733" Content="{Binding Path=DateUpdate}" />
</Border>
```

Рисунок 7 – Листинг примера привязки свойств View и ViewModel

В рассмотренном примере реализации получаем готовый шаблон, с помощью которого можно вести дальнейшую разработку. Полученный результат, показанный в рамках данного примера можем увидеть на форме. Итоговая форма с полученными данными файлов показана на рисунке 8

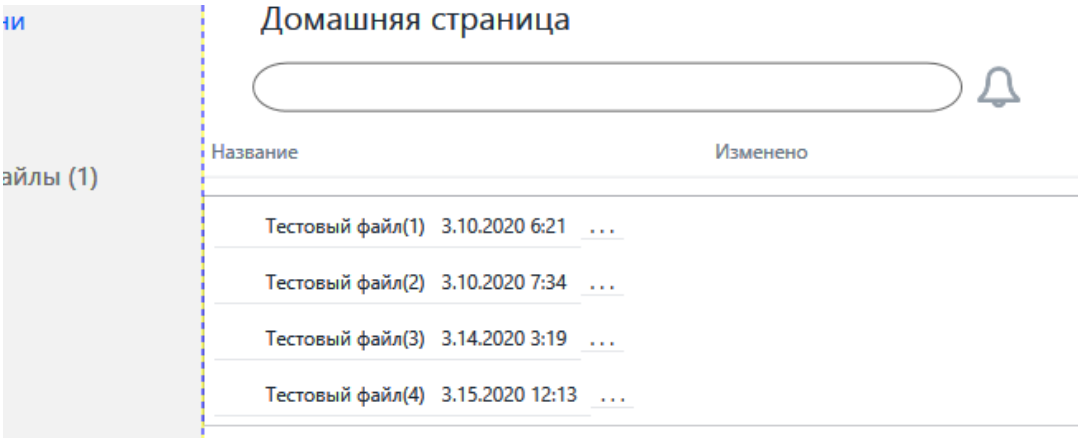

Рисунок 8 – Форма WPF с данными связанными на основе MVVM

Список литературы:

1. Документация Visual Studio. Общие сведения о WPF и XAML. [Электронный ресурс]: Электрон. Текстовые дан. – 2018 - URL: https://docs.microsoft.com/ru-ru/visualstudio/designers/getting-started-with-wpf
2. Криптография в C# (RSA, DES, SHA1). [Электронный ресурс]: Электрон. Текстовые дан. – 2015 - URL: https://studlearn.com/works/details/primery-kriptografii-na-c-rsa-des-sha1-575
3. Что такое менеджер паролей и для чего он вашей компании. [Электронный ресурс]: Электрон. Текстовые дан. – 2015 - URL: http://ru.99rabbits.com/get-password-manager
4. Donohue, Brian. Двухфакторная аутентификация: что это? [Электронный ресурс]: B. Donohue - Электрон. Текстовые дан. – 2014 – URL: https://www.kaspersky.ru/blog/what_is_two_factor_authenticatio/4272