ImageFiMan - A Windows Desktop Application for Image

Graeson Thomas, *Affiliate Member, IEEE*

Abstract—ImageFiMan is a Windows software application that helps users curate a digital family photo library that feels like the real thing but better to bring back the days of reminiscing and feeling nostalgic while browsing collections of forgotten family photos. Unlike existing options on Windows, ImageFiMan's focus is helping users find and view their personal images and to remove the current frustration that exists when trying to do so on mobile and desktop operating systems.

Index Terms—System, Computer Software, Software Prototype, Image and Video Data, Comparison and Similarity, General Public.

TABLE OF CONTENTS

• *G. Thomas was with the College of Science, Engineering, and Technology, University of Arkansas Grantham, Little Rock, AR, 72207. (email: gthomas16@uagrantham.edu)*

Manuscript received July 02, 2024; revised August 26, 2015.

1 INTRODUCTION

THE ImageFiMan application aims to evolve to an image viewing experience like that found on social media image-hosting websites such as Facebook, Instagram, and Flickr. Within a desktop Windows environment especially, creating this experience necessitates allowing the user to customize which image files appear for view operations to sort and find images.

In furtherance of this, ImageFiMan's initial focus is on managing duplicate images, including view these images and their metadata, which will allow users to manage both undesired and desired duplicate image files.

2 PROBLEM BACKGROUND

2.1 ImageFiMan Project Motivation

Since the early 2000s, digital photos have been replacing physical prints of photos to the point that the family photos of most individuals exist exclusively as digital files

stored within personal mobile and desktop computers and, more recently, cloud drive storage systems like Microsoft OneDrive. While this has made it more convenient to capture images and send them to others, it has made it harder to browse personal collections of images and to find them again later without uploading them to social media or duplicating them in a separate folder. ImageFiMan hopes to solve this problem.

The primary motivation of this project is the inadequate quality of the Windows built-in image view and the difficulty in managing image files and both intentional and unintentional redundant backups of those image files.

2.2 Problem Domain and Rationale

ImageFiMan intends to simplify the viewing and management of personal images files within a Windows desktop environment. Current options for image management are insufficient in their offerings, making it difficult to find unintended duplicate files, retain intended duplicate files, and find files on a device.

A large motivating factor is personal experience trying to manage various image files including scanned family photos of various qualities, images backed up from my phone via OneDrive sync, images backed up to my computer manually, and images copied to a third backup location. Image files are not small, and unnecessary duplicates can quickly take over available storage space. Unfortunately, available systems make it difficult to easily compare files and metadata to ensure I am retaining the copy closest to the original file even if I can find the duplicate files. Especially since images are difficult to view in the default Windows viewer.

Future intended features are intended to solve the limitations of managing images on a digital device that are not present physically. For example, use of metadata fields and display of those to create an experience like writing on the back of a photograph, and virtual organization of images in relation to one another without renaming or moving these files.

2.2.1 Definitions

- **Application Programming Interface (API)**: Protocols that make it easier for application developers to create software applications that communicate with or use the functionalities of another existing software applications [1]. For example, Microsoft Excel is part of the .NET API, which allows a developer to use Microsoft Excel functions just by adding a reference to the Excel API.
- **Archiving**: Preserving an original record and/or maintaining a record of changes to that record to retain the ability to ascertain the originating source/context.
- **Directory** (*computing*): Interchangeably referred to as a folder. A file organization structure containing references to files or to other directories (folders). In Windows, users access directories using Windows File Explorer.
- **Logic** (*computing*): An application's program code that gives the appearance of logical operations. As opposed to application program code focused on creating visual user interfaces.
- **Network attached storage (NAS) device**: A digital storage device, such as a hard drive disk (HDD) or Solid-State Drive (SSD), that is directly connected to a computer network, which allows other computers on the network to access the same files. A NAS device can be simply a HDD plugged via USB into a home network router, but some NAS devices are sold with simple operating systems to offer additional functionalities [2]. An example of the latter is the NAS I own and refer to below, a Synology DiskStation DS723+.
- **Personal files**: Files a user created themselves or directly added to their device that are usually interacted with directly. As opposed to system and application files which exist within a user's personal computer because they are needed to support the functions of the computer's operating system or to support the functions of installed software applications.

2.3 Project Market Environment

2.3.1 Current Technological Trends

Current technological trends in digital image file management include integration of Artificial Intelligence (AI) as seen in Excire Foto and integrating cloud-based services such as Bridge by Adobe, Adobe Lightroom, and Google Photo [3].

These trends are enormous for image file management, especially as file storage trends toward cloud storage over other means. However, these trends have their limitations especially as some individuals are mistrusting of AI and/or public cloud storage especially with the increasing number of data breaches and the immense of personal data that is contained within the metadata of current-day image files not to mention what might be depicted in the visual image.

2.3.2 Open Related Problems

While image file management seems an afterthought, individuals often have sentimental attachment to photos and rarely have physical copies of these images. Without proper management of these files, they can be easily lost due to device failure, or they can be easily overwritten intentionally when editing images to share with others, such as on social media. Users of social media may also be disappointed to find that the images they stored on social media are not a sufficient backup of the originals, if lost, as those images often are a much lower resolution than the original image.

Though ImageFiMan is meant to help individuals manage their personal image files, businesses also have a personal stake in managing their image files, especially those files they must maintain as business records whether for legal or proprietary purposes.

2.3.3 Recent Promising Development and Related Work

Advancements in artificial intelligence are promising, if integrated with image file management. Current AI development increasingly utilizes statistics in decision making which has allowed for systems such as ChatGPT and

generative AI that can be good at recognizing patterns and creating output based on these patterns. These AI systems can create art and music. AI is already integrated in photo viewing applications to automatically recognize and identify faces in images, create albums based on the contents of the image, and more. Utilizing AI in organizing these files could be extremely powerful, especially in finding edited versions of images and grouping them together as duplicates.

2.3.4 Existing Related Work

Existing programs that provide similar services as ImageFiMan for managing duplicate files exist such as Quick Photo Finder, CCleaner, and Tonfotos Duplicate Photo Finder Software [4].

Additional inspiration for the ImageFiMan project may be taken from the digital evidence management needs of the judicial system which require a high level of tracking and would benefit from a high level of organization and indexing [5]; and, as mentioned earlier, users' experience on social media websites focused on sharing, viewing, and managing personal images.

3 Technology Solution

The ImageFiMan project is an image file management software application for desktop devices running Windows 10 or higher intended to help individuals locate, organize, and remove undesired redundancy from their personal collection of digital images. The primary objective of this project is to provide users with a way to browse, view, and manage image files that feels more like real-life management of images with the added benefit of the ability to filter and sort images without changing their storage location or renaming files.

Accordingly, primary required user features include browsing and viewing local image files; finding, marking, and comparing potential duplicate image files; and metadata exporting, editing, and change tracking. These features allow for intended future expansion and provide users with a lightweight interface to compare potential duplicate images and select which of these to keep/delete. ImageFiMan hopes to expand upon its primary required features to provide further sorting, organization, and archiving of image files as mentioned above.

3.1 Development Plan

To develop the ImageFiMan project, I will use the Windows Platform Foundation (WPF) Application framework with .NET 8 and C# for programming language. Application development will occur in a Windows 11 operating system environment, primarily on a desktop computer.

WPF is a XAML-based UI framework that supports advanced graphics and theming required to display images and flexible user interaction including creation of multiple windows and native touch and stylus interaction support [6], [7]. I considered utilizing the application framework WinUI 3 instead, because it has advanced UI design features via Fluent design principles [6]. I decided to work with the WPF framework because, thanks to its age, it has broader support, documentation, and features [6].

While much of the solution for this project's problem will come from the vast capabilities of the .NET API, C# will enact ImageFiMan's critical business logic translating user actions into computer instructions to make decisions, update the data and image shown in the user interface, enact user changes to files, and more. Most of the written program will be in C#. While .NET applications may also be programmed using Visual Basic or F#, but C# is the most capable of those three languages and, therefore, best suited for the future growth planned for the ImageFiMan project [8].

3.2 Security Design Plan

The ImageFiMan program does not require an internet connection after installation, working locally on a user's personal computer. Therefore, ImageFiMan can mostly rely on security features built into Windows and .NET [9], including the Windows built-in role-based access-control which controls file access and modification.

Major security consideration during development include preventing unintentional or inaccurate file modification and safe handling of any user input [10]. Program development closely followed Microsoft's established security standards, which are written to align with both U.S. federal and international security standards and laws.

Through ongoing development, ImageFiMan aims to follow a model like DevSecOps by making security is part of development and written code instead of treated as a separate concern [11]). This ensures security is part of every task completed in the development and maintenance of the ImageFiMan project.

3.3 Program Performance

At this stage, little progress has been made on true user interaction, including allowing users to organize and modify their image files. ImageFiMan's initial release contains the functionality required to implement it's application environment, including a local application database, and display image files associates with a duplicate file report. While initial plans intended to include functions to organize and modify these files, scheduling issues require use to implement these operations in a subsequent feature release of ImageFiMan.

3.4 Program Deficits

3.4.1 WPF Development Framework

The Windows Presentation Foundation (WPF) development framework is older and may result in future bottlenecks, if WinUI becomes more dominant of the two or if the user experience in WPF harms application popularity. For this project, there was no clear winner between the two which may cause a problem in the future. However, the decision stands due to my lack of experience with WinUI, it is effective restriction to Windows 11, and vague concern that WinUI will be missing a third-party library necessary to create the ImageFiMan application.

3.4.2 Lack of Attractive Features

Currently, ImageFiMan's selling point is its ability to reliably display the available images, which is an improvement over Windows 11 Desktop's built-in PhotoViewier app. However, this quality is not marketable and it will be difficult to attract users.

4 Business and Technical Case

Inspiration for the ImageFiMan application comes from the developer's difficulty managing personal collections of image files, including backup copies and unintentional duplicates. Major challenges include locating duplicate image files and easily comparing both the file contents to determine which file or files to retain and which to delete. Additional challenges include organizing, labeling, and relocating retained images files to make their usability closer to that of a family photo album.

While Windows 11 has a built-in image viewer, it has several bugs that make it difficult to use including the fact that, when viewing images stored on OneDrive, the photo viewer resets the current image when OneDrive syncs. Additionally, Windows 11 provides limited access to metadata within an image, making it harder to determine the original file out of a group of duplicate files.

The closest existing solution to image organization is social media photo sharing, which allows people to organize, tag, and provide descriptions of these images. However, they do not retain the original files and require the user to sacrifice privacy. This is a core motivation for creating a desktop application to provide this experience on a user's local computer even though ImageFiMan's initial focus will be on managing duplicate images with future goals of expanding to provide advanced image organization and viewing capabilities.

Considering the problem environment of a Windows 11 operating system on a desktop computer, options for solution delivery cannot include internet or web-based applications because they expose the user to privacy risks that ImageFiMan wants to avoid, including that of transmitting personal information over the internet. The metadata on most images taken today contains extensive personal information of the person who captures that information, including the GPS coordinates of where the image was captured, the capturing device's information, and even the device owner's name.

The developer had previously started work on a similar project using Python, a program language used for scripting or command line programming, which made it one option for solution delivery. The benefits of this option include the benefit of familiarity of developing programs with Python, the partial work already completed towards the intended goal, and its flexibility during application development. Downsides of this approach include Python's slowness, difficulty creating desktop graphical user interfaces, and its lack of streamlined packaging for delivery. Largely, Python programs are released as web applications or as packages installed using the built in command line tool, Pip. This reduces the audience for the application.

Another option for delivery is an executable or via the Windows Store, both of which are the most common method of delivery for Windows desktop applications. Current development for Window Store application is via Win UI, which is a newer development framework. While releasing the application on the Windows Store would make it most accessible and more likely to gain user support, Windows Store applications have limited access to user's files which is a requirement of the ImageFiMan application to be able to manage user's image files. Therefore, this option could not be pursued.

The final delivery option mentioned above is via an executable file. This option can be easily created when developing an application using .NET or .NET Framework, both of which are native to Windows. Additionally, this option is as easy for users to install as a Windows Store application, though users are, rightly so, more suspicious of executable files, because they can be used to install malicious programs such as spyware, malware, and Trojans. In other words, the largest downside of this option is the increased difficulty of gaining user trust to install the application.

Considering available options, delivery by executable is recommended. While users have become more cautious about executable files compared to Windows Store applications, they are still vastly utilized to release software, including to individual users and corporations. This option is far more accessible than delivery via a package manager like that used by Python. Since the application will not function properly if delivered via the Windows Store, this leaves only the executable option.

To enact this delivery option as a native Windows application, we will use C# and .NET Core to develop the application, which can be easily packaged and released as an executable file including via GitHub.

5 Design Methodology

Since ImageFiMan will be a Windows 11 desktop application developed using C# and .NET, design will be accomplished using the Model-View-ViewModel (MVVM) theory of application structure, in which the Model represents data from a file store, the View manages the user's view of the application, and the ViewModel handles the interaction between the Model and the View, similar to the Controller part of the Model-View-Controller design method utilized in .NET web applications [12]. This method provides separation of duties, which makes it easier to respond rapidly to changes and provides a foundation for future expansion because it allows for incremental changes without completely rehauling the application.

Inspiration for the user interface was also taken from the image viewing application in my Network Attached Storage system, which provides a more thorough view of image metadata; see an example below in Figure 1.

As mentioned above, initial development is limited to managing duplicate files. Figure 2 depicts ImageFiMan's program design schema.

Figure 2 depicts how the program generates an image file database from file(s) on the user's system, displays the collected information to the user who can change the display of the images and save those changes to the database and/or the image files themselves. Changes of view include

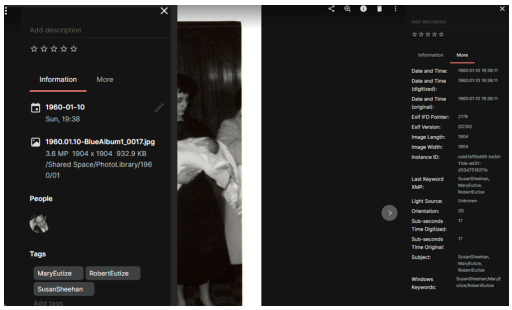

Fig. 1. NAS metadata output, showing the two formats available for one image

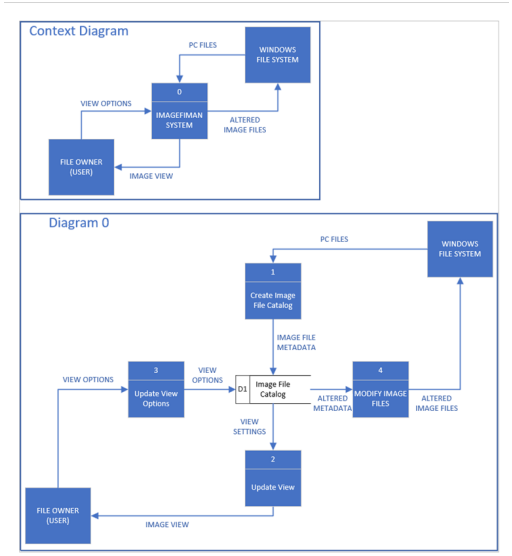

Fig. 2. ImageFiMan's Data Diagram

marking images for deletion or hiding, editing image metadata, and sorting images. Some of these changes such as sorting are saved in the database, which allows the addition of features not available in Windows or the image file structure itself.

Focusing on these core features will make the application less competitive initially but will allow us to quickly release a function application which we will then grow into a more robust and feature-rich application.

6 Final Design

ImageFiMan is a desktop Windows application implemented using the programming language C# and .NET 8. Microsoft's .NET is an extensive code library, accessible via API, that simplifies application development by creating a common system that can be used across various operating systems and devices [13]. For example, developers can use .NET's System.IO module to read and write files, without worrying about the end users' operating system. The immense benefit of .NET, especially within Windows application development, made it a necessary component of ImageFiMan, which subsequently led the decision to write ImageFiMan in C#.

While the .NET API can be leveraged by other programming languages [14], C# was created by Microsoft to work within the .NET Framework [13], which is the predecessor to newer .NET. Additionally, C# is a powerful and popular [15] object-oriented programming (OOP) language, with a plethora of online support resources including documentation, guides, and examples. These resources are invaluable during application development and C#'s OOP structure is crucial to maintaining code organization throughout an application's life cycle, especially in a growing project like ImageFiMan.

6.1 Code System Details

The ImageFiMan code system is organized using a Model-View-ViewModel (MVVM) architectural pattern, in which code is grouped based on into three structures – the Model, View, and ViewModel – each of which have separate responsibilities. The Model represents data and contains business logic, the View generates the display shown to users, and the ViewModel handles the interaction between the View and Model. ImageFiMan's core architecture is comprised of two Views, MainWindow.xaml and PhotoViewer.xaml; several Models including DuplicateReport, DuplicateGroup, DuplicateFile, Photo, and ExifMetadata; and the ViewModels: MainWindow.xaml.cs, PhotoViewer.xaml.cs, PhotoCollection, and DuplicateFileContext.

App.xmal provides application-wide definitions including setting the application's entry point (i.e., startup URI) as MainWindow.xaml and defining the application-wide resource, "Pages," as a data binding source (i.e., object data provider) of type PhotoCollection. Data binding sources are the connecting points through which the View and ViewModel communicate about business objects. Accordingly, the PhotoCollection ViewModel object uses Models, including the DuplicateFile and DuplicateFile models, to serve duplicate group and duplicate file information to the user.

On application startup, program execution creates a MainWindow. Prior to its presentation to the user, the MainWindow utilizes the App.xmal-defined Photos resource and the PhotoCollection ViewModel to create and set the Photos binding between the MainWindow's ViewModel and its View. In MainWindow.xaml.cs, this includes initializing an empty list to store DuplicateFile objects. In MainWindow.xaml, this binding is set via the Grid's DataContext property. While the Photos communication channel exists at this point, it does not contain data and has nothing to serve to the MainWindow View. If the Main window were to load at this point, it would be empty of data, as shown in Figure 3.

Fig. 3. The ImagFiMan, showing an empty three-panel layout

Instead of depicting an empty window, the MainWindow's ViewModel (in MainWindow.xaml.cs) obtains the Photos collection data from ImageFiMan's application database by creating a database session, represented by the DuplicateFileContext class as defined in DuplicateFileContext.cs and its superclass, the Microsoft Entity Framework Core's DbContext class.

6.1.1 Model-ViewModel Interaction

The Microsoft Entity Framework Core (EF Core) .NET module functions as an object-relational mapper between databases and .NET objects [16], and its DbContext class encapsulates the common logic required to generate a database and a connection to that database from program code defined using data models and a custom class derived from the modules DbContext class [17].

Accordingly, the code within DuplicateFileContext.cs provides the customization necessary for the DbContext superclass to operate. Specifically, upon creation in MainWindow.xaml.cs, the DuplicateFileContext object creates a local application folder for itself, if that folder does not already exist, generates a file path to the application's database, and makes that file path accessible via the DuplicateFileContext object's DbPath. Immediately after object construction, the DuplicateFileContext object's OnConfiguring method handles the raised DbContext's Configuring event, to configure the database provider to SQLite, the database's data source to the DbPath location, and to enable lazy loading of child rows as they are requested.

While EF Core can generate and interface with the ImageFiMan database without further customization, the DuplicateFileContext object overrides EF Core's DbContext's OnModelCreating method to customize the database primary key for the DuplicateGroup entity, add Alternate Keys to the DuplicateGroup and DuplicateFile entities, and create an index on the DuplicateFile entity's filename field.

Additionally, within DuplicateFileGroupContext's OnModelConfiguring method, the database is populated, if necessary, using the ModelBuilder.Seed extension model, defined in DuplModelExtensionBuilder.cs, to add initial data from a duplicate file report.

C#'s extension method allows a developer to add a method to an existing .NET class without creating an inheritance structure as used for DuplicateFileGroupContext [18]. In other words, the code in DuplModelExtensionBuilder.cs adds the Seed method to EF Core's ModelBuilder type without altering or overriding the ModelBuilder class or changing the default value passed to the OnModelCreating method.

Within this Seed extension method, the database's initial state is generated from an externally generated duplicate file report, in which each line corresponds to image file paths with tab-separated values including each file's group number, which groups potentially identically images. In each report file, the group number starts at 1 and increases by one with each subsequent group.

To uniquely identify duplicate groups across duplicate reports, the program code within the Seed extension method assigns each group a unique DuplicateGroupId primary key to simplify entity relationships and uniquely identifies incoming data via a composite alternate key made of the report's original group number and the foreign key, DuplicateReportId. This composite key allows users to import subsequent file reports and add only groups that were not already added, whether from a prior report or from a new report.

Using this importing algorithm has the added benefit of being extensible, and it will serve as the foundation for future development iterations including more intelligent recognition of groups that do match across duplicate file reports and to allow users to modify the members of these groups, including combining and merging groups.

After adding the DuplicateReport data to the database, the Seed method reads each line from the report, parsing the DuplicateFile data from each line, and associating each with its corresponding parent DuplicateGroup. Microsoft Entity Framework creates a database matching the program models.

Finally, the MainWindow ViewModel obtains a list of DuplicateGroups from the database and passes that list to directly to the ImageFiMan's main navigation pane, a TreeView control identified by the name trvFileGroup.

6.1.2 View-ViewModel Interaction

Unlike the Photos databinding described above, the MainWindow ViewModel passes the groups list directly to the MainWindow View. This type of binding is less dynamic, and more appropriate for the TreeView data, which presently remains unchanged throughout program runtime. In MainWindow.xaml, the TreeView's parent controls set its positioning to the left-side of the ImageFiMan's main window and the TreeView XAML sets the event handler for its SelectedItemChanged event, which is raised when the user selects a different TreeViewItem.

However, the TreeView does not appear to contain any Items. Instead, the TreeViewItem objects are dynamically generated according to the DuplicateGroup DataTemplate defined in the MainWindow's Window.Resource. This DataTemplate describes the visual representation of each DuplicateGroup, represented as its original group number followed by the number of child files within it.

When the user selects one of these groups in the TreeView, this interaction raises the TreeView control's SelectedItemChanged event and the MainWindow ViewModel's OnTrvSelected method handles the raised TreeView event, setting the PhotoCollection binding's DuplicateFile list to the selected group's DuplicateFiles list. On change to the PhotoCollection DuplicateFiles list, populates itself with Photo objects created from each DuplicateFile in the list.

The Photo object stored in the PhotosCollection are defined in ImageFiMan Photo.cs to represent a system image file, including its metadata in two formats. One metadata format, accessible via the Photo object's MetaTags ICollection, consisting of metadata tags and their descriptions used to display selected image metadata in in the MainWindow and PhotoViewer's left-side pane. The other format, in the Photo's Metadata provides a datetime label for each image preview displayed in the MainWindow's ListBox.

To create each Photo object, the PhotoCollection accesses the system file using the DuplicateFile object's file path to access the image data, which is converted into

an in-memory image frame using the .NET module System.Windows.Media.Imaging's BitmapFrame class. A third party Nuget package, MetadataExtrator simplified extraction of metadata from the system image file, generating a dictionary-like directory of metatags grouped by metadata source. Initially use of this directory resulted in inconsistent results depending on the original image file format. Therefore, a stop-gap measure was implemented via the Photo object's two-version metadata.

First, the Photo's ExifMetadata objects iterates through the metadata retrieved using MetadataExtrator. During each iteration, ExifMetadata searches for metadata tag names containing matching the key words, "width," "height," and "datetime" and saves them to corresponding private lists. If these lists are not empty, the ExifMetadata object's corresponding public properties return the first metatag from the list. Secondly, the MetaTag list is populated by flattening the MetadataExtrator metadata, removing the directory grouping and any metatags with descriptions longer than twenty characters.

While users may easily compare potential duplicate images and view their metadata within ImageFiMan's MainWindow, users can double click on a preview image to open it in a new PhotoViewer Window. The MainWindow ViewModel's OnPhotoClick method creates the new PhotoViewer with its SelectedPhoto property set to the MainWindow ListBox's selected photo item. This Photo object remains the same even if the user selects a different image in the MainWindow. Through this command, users can create multiple PhotoViewer windows that make it even easier to compare duplicate images and locate any differences between their metadata.

7 Solution Implementation

The ImageFiMan solution is implemented in a field environment, executed using local system files, established on a user's system after the user runs ImageFiMan's executable installation file. During this installation process, Microsoft Entity Framework Core's database migration feature establishes the program's database based on three program models, DuplicateReport, DuplicateGroup, and DuplicateFile, the data design schematic as shown in Figure 4.

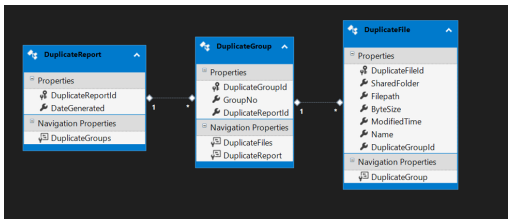

Fig. 4. Data Design Schematic depicting the DuplicateReport, DuplicateGroup, and DuplicateFile objects

The DuplicateFile model's unique property is its file path, which is the computer file system location of that DuplicateFile's image file. In ImageFiMan's implementation, the Photo model represents this image file, conveying the visual contents of the image and the image's metadata to the user through the MainWindow and PhotoViewer Windows.

Program execution within ImageFiMan starts in the MainWindow's ViewModel with the creation and seeding of the application's database, if necessary, the creation of in-program representations of that data, and the presentation of that data to the user. An overview of ImageFiMan's program execution can be seen in Figure 5.

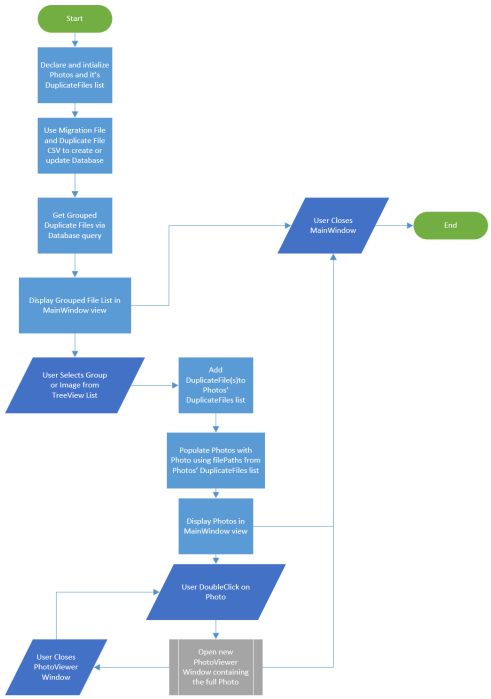

Fig. 5. Program design schematic showing program execution,

7.1 Implementation Challenges

7.1.1 Establishing Program Environment - Persistent Data Store

To establish the foundation of the program, I needed to establish a database from a text file containing tab separated data. I had no experience creating a database outside of Python programming. Repeated attempts failed either to establish the database or to properly interact with that database. When these failures resulted in exceptions or errors, they were related to foreign key/entity relationship issues, which were supposed to be automatically managed by Entity Framework Core. I was not able to diagnose the problem with other databases that were created, but this was resolved after repeated attempts. In the last database creation, the problem was loading the files. All the official examples that I followed did not show manually loading the related/navigable field (i.e., the duplicate groups duplicate files list). Including this operation (the Include method) fixed part of the issue.

There are no outstanding challenges or obstacles to this challenge. The database will need to be expanded as the program grows, but the program could not move forward without the process of establishing the initial database and seeding it with data.

7.1.2 Display Images

This challenge involved a user interface adjustment and event handling. My initial intention was to create a main window from scratch using a TreeView, but I found myself unsure of how to proceed forward after creating the TreeView of grouped files. I had less than 24 hours to create a prototype solution; therefore, I rewrite/copied core portions of code from an example photo viewer application from a Microsoft Learn GitHub Repository [19], which I adjusted to fit my needs.

The next large challenge here was figuring out how to display images in the Main window. The example program uses a single directory containing a handful of example files, but the grouped files I wanted to display are in different folders. This I solved using a private list of DuplicateFiles within the PhotoCollection object. When the user selects a DuplicateGroup, the PhotoCollection's DuplicateFiles list is set to group's DuplicateFiles list. If the user selects a DuplicateFile, then that file is added to the PhotoCollection group.

Prior to this, I had to figure out how to add an event handler to the TreeView items when they do not exist as individual objects in the XAML or C# code. Through this process, I realized I could view control Properties when they are in focus in the XAML code. Before this, I thought I had to use the Design view to access this. This made it far easier to find the correct Event since I could easily filter the list to just events, view tooltip explanations, and move the OnTrvSelected event handler to different Events when using trial and error process to find the one matching my desired user action.

While fully function, the display image(s) feature needs improvement and optimization, both to the TreeView navigational menu and the Photo Viewer windows. Current ideas for future approaches include a list of groups with a small preview image to represent the contents of the group or replace the image previews with the groups and the PhotoViewer window with something like a Slideshow viewer that allows one to flip through the images within the group in a "carousel" or circular loop. The downside to the latter option is the loss of the ability to view the duplicate images "side-by-side."

This outstanding issue must be fully addressed in later releases, in which I intend to incorporate ViewModel objects that are user-side versions of duplicate groups and duplicate files. For example, the user-side DuplicateGroup may contain multiple DuplicateGroup objects or a subset of DuplicateFile objects from one or more DuplicateGroup objects. The user-side versions will also exist in the Database to remain persistent over multiple sessions.

7.1.3 Display Image Metadata

A key feature of duplicate image file management is the ability to compare image file data to determine which file to retain and which to delete, if desired. Currently, the metadata display is inconsistent, retrieving data only from the Exif metadata group when not all types of image files store data in this group. The image files used by ImageFiMan include jpg/jpeg and png files. Some were created on digital cameras, some on cellphones, and some input via digital scanners from physical images, contributing to this difficulty.

The MetadataExtractor by Drew Noakes helped solve this problem, as described in the Final Design section. However, ImageFiMan's current implementation repeats similar operations to on every image displayed, creating excessive and unnecessary overhead. This is my top priority to resolve in the next release of ImageFiMan.

7.1.4 User File Changes

This challenge was caused by all those listed above, which prevented work on this feature. This feature was intended as part of the initial release of the program but seems unlikely to be possible. This was a concern during initial planning, which is why initial work still provides a foundation for these features. The main negative impact is loss of competitiveness as, without these features, the initial release will only display images.

Mentioned above, the future approach will be to add ViewModel class objects that filter, sort, and group the database model objects according to the user changes without altering any non-program system files. This approach mocks test driven development while also providing a platform for future growth as file system changes change stored in the database can be enacted in the system after confirmation by the user. In other words, when we want to add the feature to enact system changes, the ViewModel objects will function as a list of pending changes that will only be enacted after final confirmation. Some changes, however, will never alter the system files as they are only meant to change the user's display of the images and not the image files themselves.

7.2 Program Performance

Currently the program takes too long to startup loading all image groups and their related files into the tree view menu. Additionally, the user interface has unnecessary features and poor user experience which leads to inferior performance. The image viewing functionality is fast and responsive thanks to its light weight. Better user experience could allow the program to still be preferable over the built in Windows image viewer, because the current program does not unexpectedly change your current image to the first listed in its directory.

The current implementation does not perform as planned, barely functioning as an image viewer. Work on the project was severely delayed due to the developer's health issues, further compounded by the developer's inexperience with establishing a local database using Entity Framework. These delays have led to the current solution, which is in its current state due to lack of time spent on development.

At present, the only security measures are those built into .NET and the Windows operating system, which limits file access and modification based on built-in user access controls. At present, there are no additional security concerns as the program only alters its own program data and reads local file data.

The biggest negative influence on performance is the loading of DuplicateFile Groups and all their DuplicateFile

objects on program load. This is especially true since, in its current state, it is unnecessary to list the files within a group. In other words, the current performance is especially bad because we do not have a reason to load all objects. Addressing this performance issue will require a redesigned user interface and may be too big of a task for the initial project's timeline. However, it should be a major focus of the second version of this program.

The performance impact of unnecessary features and poor user experience with the current user interface must be resolved before final release and, therefore, will not be discussed in detail. The fixes are minor and the biggest barrier to their resolution is distraction on secondary, non-critical features. Particularly, the one just mentioned as the TreeView is part of the poor user interface but meeting the project deadline will require overlooking that beyond visual appearance and more minor but meaningful user experience design choices such as ensuring proper tab order and accessibility.

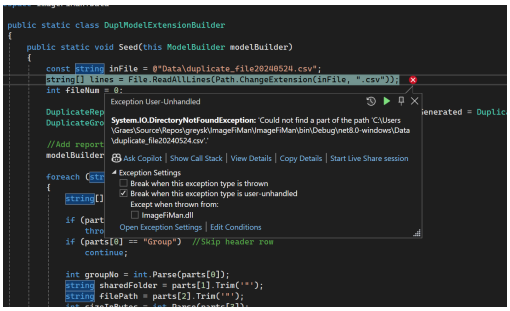

Fig. 6. Exception message experienced when adding smaller dataset

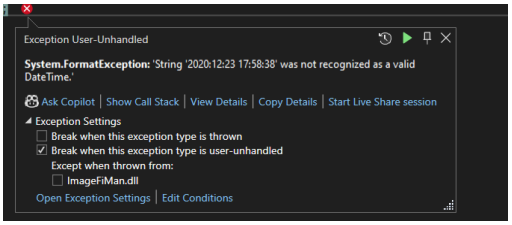

Fig. 7. Exception message experience trying to display image datetime metadata

8 Evaluation of Solution Effectiveness

8.1 Programmer Test Plan

ImageFiMan testing occurred throughout program development after each major modification. Additional user testing occurred during deployment, and major bugs corrected before application release.

Both testing stages uses two datasets, an example dataset included in the release application, and a dataset built from my person image file collection and a duplicate file report generated by my network attached storage.

I decided to test the application on the data set, generated from my personal image files, because ImageFiMan was inspired by this image file collection and because the data was readily available. While this large dataset benefited initial application development by quickly identifying performance bottlenecks, it's size made it impossible to port for testing outside of my personal desktop computer.

Therefore, I created an example dataset comprised of two groups and four images from the original dataset of my personal image collection. The release of ImageFiMan includes this data subset to provide initial, example functionality for both end users and program testers. This smaller subset allowed me to test database integration upon first program deployment, because I could easily drop and recreate the local database as well as install and test the application on other Windows computers.

8.1.1 Test Case Scenarios

During application development, I primarily applied three test case scenarios to validate correct generation of the visual user interface, expected response to user interactions, and accurate program data state to generate expected results.

For visual user interface testing, I employed manual visual inspection testing and end user testing in addition to the development tools built into Visual Studio 2022.

Program data testing was completed by manually verifying correct output and addressing exceptions generated during program development. One such exception, shown in Figure 6, arose when adding the smaller data set to the programming test plan. Instead of seeding the database with the duplicate file report information, the program could not find the report file. This error occurred because I failed to include the report file in the program's executable files.

When responding to users feedback to correct image metadata, another error arose because C# could not recognize image date time metadata descriptions having date values separated by colons.

This error, shown in Figure 7, occurred when the program attempted to display the image's date in its preview image label or in the Main Window metadata sidebar. Outside of debug mode, this exception caused the program to crash.

8.1.2 Potential Legal, Ethical, or Security Issues

There are currently no potential legal, ethical, or security issues detected or potentially arising from the ImageFiMan software.

I am committed to open and honest communication regarding all future ImageFiMan development, including remaining cognizant of any legal, ethical, and/or security issues that may arise during future development and during the application's maintenance life cycle and notifying users and potentially impacted parties shall I detect or foresee these issues.

8.2 End User Test Plan

8.2.1 User Experience

During end user testing, users were quickly able to interact with ImageFiMan's Main Window to preview images using the TreeView side navigation. However, users were frequently unaware of the ability to open images in a new window. When users did open photo view windows, they initially enacted this operation accidentally.

While the ImageFiMan program functioned as intended, end users did not know how to interact with the program as developers expected. When users became confused, the

application provided no help, because it currently lacks a help utility or user documentation. This contributed to the poor user feedback received during testing.

8.2.2 User Feedback

All users were dissatisfied with the ImageFiMan's functionality, specifically its lack of features.

During initial end user testings, users were frustrated and confused by ImageFiMan's irreliable display of image metadata and by the file names in the TreeView side navigation pane. User feedback overall was to limit the TreeView navigation to group items only and fix the image metadata.

Users' responses to ImageFiMan were understandable, especially during initial testing when the application barely and unreliably displayed image file data. I expected disappointment from users, because the delivered application fails to meet the majority of requirements as outlined during the system analysis phase of application development.

I hope to use future releases to deliver feature improvements to end users, including those necessary to live up to ImageFiMan's design plans and meet end user requirements for desktop image file management.

9 Improvements and Optimizations

9.1 Programmer Test Plan Improvements

Prior to ImageFiMan's initial release, I addressed all errors found during the programmer's test plan. The two major bugs described in the prior section were corrected with simple code fixes.

Major outstanding issues include ImageFiMan's lack of features, the subpar user interaction experience, and the static nature of its state data. Future program revisions will address these major issues by implementing the display of users' personal image files, the ability for users to modify the groups and files displayed in the Main Window, and to alter image metadata.

Minor outstanding issues and bugs include the lack of optimization within ImageFiMan. An obvious example are the redundant metadata lists contained within the Photo class. In future releases, I will utilize a single list, moving the Width, Height, and DateTime metadata properties to the Photo object.

9.2 End User Test Plan Improvements

The most concerning result of end user testing is the user's lack of interest in the current solution. While this result is understandable given ImageFiMan's current lack of features, this result inhibits any future market growth for ImageFiMan. This feedback is a critical issue that must be addressed at the earliest possible opportunity.

While not a concern of end users, development of the users' requested features requires integration of automated program tests to ensure correct operation, especial when altering computer image files and information stored in ImageFiMan's database. Unfortunately, time did not allow for me to address this crucial issue prior to initial deployment.

Beyond requesting additional features, users were unhappy with the initial user interaction and interface. Of particular concern was the display of unnecessary information, especially interface items that implied the existence of features not currently available.

Prior to subsequent improvements, the TreeView navigation panel in ImageFiMan's Main Window displayed duplicate image file names nested within their parent duplicate groups as shown below in Figure **??**.

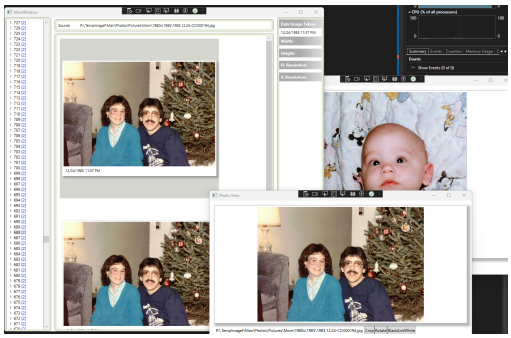

Fig. 8. The first working ImageFiMan prototype

When a user selected an item from the TreeView navigation, the image list pane would display either the single selected image or all the images in a group. Users thought it was pointless to be able to view a single image from a group within the Main Windows without the ability perform customized selections to view images across multiple groups. Additionally Users were frustrated by ImageFiMan's inconsistent display of image metadata.

I address this end user feedback by removing the nested file names from the Main Window's the TreeView navigation and establishing algorithms to reliably display metadata and a data time from all image file types. While relatively minor, these improvements satisfy ImageFiMan's core business requirement of displaying images and their metadata so users may view and compare these images.

9.3 Future Revisions

ImageFiMan's planned future revisions include separating database migration program logic from the main program logic, adding automatic unit and integration testing, and adding the ability for users to customize the application's list of image files and groups.

As ImageFiMan improves, I will tweak the user interface to improve user experience. Of utmost importance are the addition of user help facilities to help users.

10 Conclusion

In conclusion, ImageFiMan's initial release was a failure with end users but a developmental success given the short development time line and it's potential growth through future revisions and application releases.

11 References

References

[1] M. Goodwin, *What is an API (application programming interface)?* en, https://www.ibm.com/topics/api, Accessed: 2024-5-14, Apr. 2024.

[2] T. Brookes, *What is a NAS (network attached storage)?* en, https://www.howtogeek.com/742893/what-is-a-nas/, Accessed: 2024-5-14, Aug. 2021.
[3] D. Ginn, *10 best photo management software in 2024 [photo organizers]*, en, https://www.cloudwards.net/best-photo-management-software/, Accessed: 2024-5-14, Apr. 2024.
[4] Monil, *29 best duplicate photo finders & cleaners tools in 2024*, en, https://www.techpout.com/best-duplicate-photo-finder-and-cleaner-software/, Accessed: 2024-5-16, Apr. 2024.
[5] Axon, *How do digital evidence management systems work?* en, https://www.axon.com/resources/how-do-digital-evidence-management-systems-work, Accessed: 2024-5-14, May 2023.
[6] A. A. Craft, *Choose the best application framework for a windows development project*, https://learn.microsoft.com/en-us/training/modules/windows-choose-best-app-framework/, Accessed: 2024-5-14, Jan. 2024.
[7] Q. Radich, *Overview of framework options*, https://learn.microsoft.com/en-us/windows/apps/get-started/, Accessed: 2024-6-15, May 2024.
[8] B. Wagner, *.NET managed languages strategy - .NET*, https://learn.microsoft.com/en-us/dotnet/fundamentals/languages, Accessed: 2024-5-20, Feb. 2023.
[9] I. Evangelist, *Secure coding guidelines*, https://learn.microsoft.com/en-us/dotnet/standard/security/secure-coding-guidelines, Accessed: 2024-5-14, Sep. 2021.
[10] I. Evangelist, *Security and user input*, https://learn.microsoft.com/en-us/dotnet/standard/security/security-and-user-input, Accessed: 2024-5-14, Sep. 2021.
[11] RedHat, *What is DevSecOps?* https://www.redhat.com/en/topics/devops/what-is-devsecops, Accessed: 2024-7-2, May 2024.
[12] M. Stonis, *Model-View-ViewModel - .NET*, https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm, Accessed: 2024-7-2, May 2024.
[13] B. Perkins, J. V. Hammer, and J. D. Reid, *Beginning C# 6 programming with visual studio 2015*, en. Standards Information Network, 2015.
[14] G. Warren, *Language independence and language-independent components*, https://learn.microsoft.com/en-us/dotnet/standard/language-independence, Accessed: 2024-5-28, Dec. 2022.
[15] K. Dollard, *Update to the .NET language strategy*, en, https://devblogs.microsoft.com/dotnet/update-to-the-dotnet-language-strategy/, Accessed: 2024-5-28, Feb. 2023.
[16] A. J. Vickers, *Overview of entity framework core*, https://learn.microsoft.com/en-us/ef/core/, Accessed: 2024-5-28, May 2021.
[17] A. J. Vickers, *DbContext lifetime, configuration, and initialization*, https://learn.microsoft.com/en-us/ef/core/dbcontext-configuration/, Accessed: 2024-5-28, Feb. 2023.
[18] B. Wagner, *How to implement and call a custom extension method - c#*, https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/how-to-implement-and-call-a-custom-extension-method, Accessed: 2024-5-28, Mar. 2024.
[19] Microsoft Learn Contributors, *WPF-Samples: Repository for WPF related samples*, https://github.com/microsoft/WPF-Samples, Accessed: 2024-6-11, Jul. 2024.

12 Appendix

12.1 Supplemental ImageFiMan Screenshots

Fig. 9. ImageFiMan on load with large testing dataset

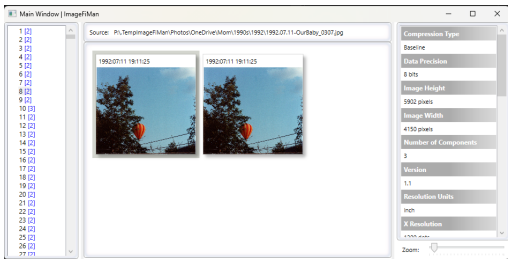

Fig. 10. ImageFiMan's Main Window after group selection

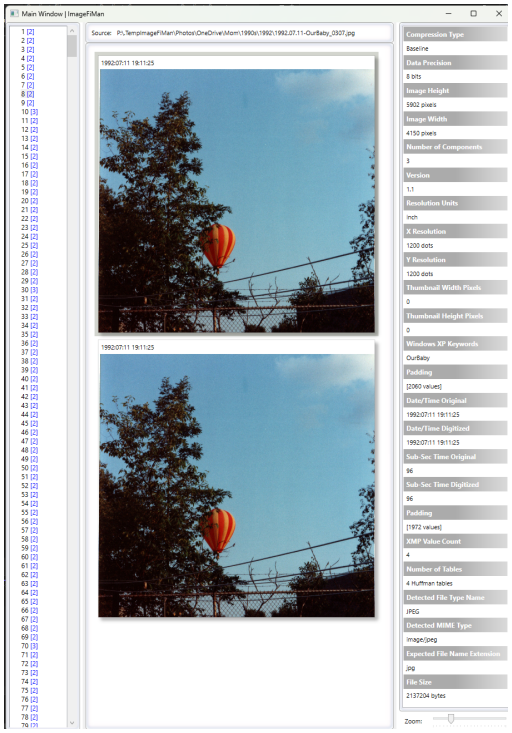

Fig. 11. ImageFiMan's Window expanded to show full list of metadata and image list zoom feature

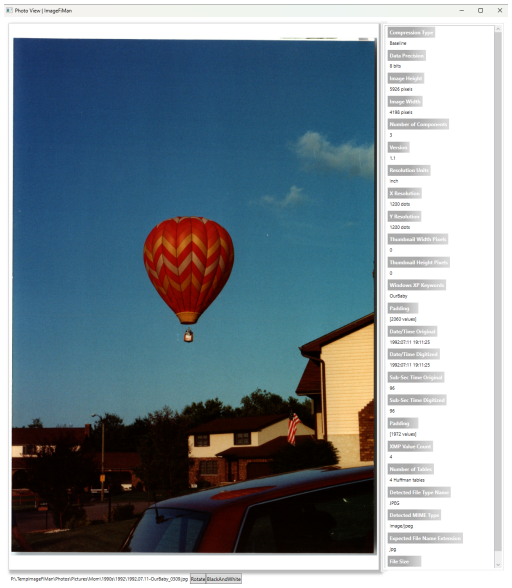

Fig. 12. ImageFiMan's Photo View Window showing full metadata list