University of Alabama in Huntsville

LOUIS

Honors Capstone Projects and Theses Honors College

4-19-2024

Dynamic Slideshow Creation Software

James Thomas Haywood
University of Alabama in Huntsville

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
Haywood, James Thomas, "Dynamic Slideshow Creation Software" (2024). *Honors Capstone Projects and Theses*. 888.
https://louis.uah.edu/honors-capstones/888

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.

Dynamic Slideshow Creation Software

by

James Thomas Haywood

An Honors Capstone

submitted in partial fulfillment of the requirements

for the Honors Diploma

to

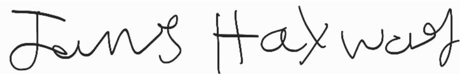
The Honors College

of

The University of Alabama in Huntsville

4/19/2024

Honors Capstone Project Director: Dr. Dan Schrimpsher

 4/22/2024

____

Student Date

4/25/2024

____

Project Director Date

____

Department Chair Date

____

Honors College Dean Date

Honors College

Frank Franz Hall

+1 (256) 824-6450 (voice)

+1 (256) 824-7339 (fax)

honors@uah.edu

Honors Thesis Copyright Permission

This form must be signed by the student and submitted with the final manuscript.

In presenting this thesis in partial fulfillment of the requirements for Honors Diploma or Certificate from The University of Alabama in Huntsville, I agree that the Library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by my advisor or, in his/her absence, by the Chair of the Department, Director of the Program, or the Dean of the Honors College. It is also understood that due recognition shall be given to me and to The University of Alabama in Huntsville in any scholarly use which may be made of any material in this thesis.

James Haywood
____

Student Name (printed)

____

Student Signature

4/22/2024
____

Date

Table of Contents

- Abstract…………………………………………………………………………………3
- Introduction……………………………………………………………………………..4
- Chapter 1: Front End Design…………………………………………………………...5
- Chapter 2: Playing A Slideshow………………………………………………………12
- Chapter 3: Back End Design…………………………………………………………..16
- Conclusion……………………………………………………………………….....21

Abstract

The purpose of this project was to create two standalone Windows programs that allow the user to create a personalized slideshow complete with dynamic transitions and music. The project was split into two smaller programs: one for creating a slideshow and one for viewing the creation. The programs were written in Python by my team and I for our CS 499 senior design class. As I was the only honors student in my team, I decided to take on extra work and implement more features into the program to make this a proper capstone project. In addition to fulfilling the original program requirements I have designed and implemented a database used to store the created slideshows. In this paper I will go over the general design process and implementation of the systems we used to create the aspects of the program such as the slide transitions and audio playback. Additionally, beyond just explaining the design and implementation I will compare the pros and cons of using a database for the purpose of storage vs just saving the data to a file and how this database system could be used to expand the original scope of the project even further.

Introduction

Have you ever wanted to show off your pictures in a personalized slideshow? Complete with a soundtrack and animated transitions? That is exactly why my team and I chose the dynamic slideshow software assignment for our senior design project. The goal of this project was to create software that allows the user to create personalized slideshows using images of their choice. The project was split into two parts: a slideshow creator program and a slideshow viewing program. Within the creator program users would be able to add JPG or PNG images into the project from anywhere on their computer, select any of those images to use as a slide, and then customize aspects of the slide such a transition, transition speed, and how long it gets displayed. Additionally, mp3, wav, or aiff music/sound files could be added to the slideshow as a playlist. Users would also be able to toggle various settings about the slideshow like if it changes slides automatically or manually and the ability to shuffle the slide/song order. Then once the user has saved their slideshow, they can use the viewing program to display their personalized slideshow.

Both parts of the program were written in Python for a Windows system. They were designed to have no dependencies and be lightweight. To make this project capstone worthy, I have taken on extra work and extended the scope of the project by a bit. Originally, the program just saved slideshow projects to a JSON format. For the capstone part of this project, I have designed and implemented a database that will house all of a user's saved projects instead of storing them in loose files. Since I am the only member of my team taking part in this capstone project two near identical versions of the programs were made: one that saves to the JSON format and one that saves to the database. In addition to just designing and implementing the database I will be comparing the pros and cons between the two saving methods and how the underlying design can be used in other scenarios.

Chapter 1: Front End Design

To start the project, we decided to go ahead and design the front-end user interface because we believed understanding what we wanted the end product to look like would allow us to better plan a path getting there. It was also very necessary as both programs are extremely front end heavy so to go forward and implement many of the features the UI had to be in a mostly complete state. This chapter will detail how the UI for the slideshow creator and player were designed and implemented along with reasoning behind many of our decisions.

GUI Tools

The slideshow software is written in Python, and in Python there are many ways to go about building a GUI. Since we knew the programs were going to be stand-alone Windows programs, we discarded the options that required the use of a browser. Additionally, we wanted to keep things simple because we did not have much experience in creating GUIs. After some deliberation and research, we boiled our choices down to be between Tkinter, PySimpleGUI, Kivy, and PyQt. In the end we decided to use Tkinter because it's included standard with Python, the large amount of information available on using Tkinter, and the fact that Tkinter has a lot of add-ons we could use to enhance it. One such add-on that we decided to use was ttkbootstrap which is a wrapper for Tkinter that helps update the kind of dated look Tkinter normally has to a more modern look. Additionally, ttkbootstrap comes in with many built-in features such as many-colored themes and some useful widgets such as a table view that we made great use of.

Creator

When designing the GUI for the creator we wanted to keep it simple and intuitive. For that reason, we took heavy inspiration from the sort of standard UI a lot of video editing software uses (e.g., iMovie, Adobe Premiere, etc.). Effectively the program window was broken up into 4 sections, one for each corner. Each section would house a specific widget that fulfills the functionality of that section. The top left corner of the program would be a media bucket which houses all the image files that the user has access to create their slideshow. The top right would be where the user can see a preview of the images. The bottom right would be a sort of slide reel where images used in the slideshow and their order would be displayed. The bottom left would be where information about the project, slide, or image would be displayed. Nearly all widgets used in the program would be custom made and put into a separate file called Widgets.py.

The media bucket widget basically handled adding files to the project and displaying the current project files. Files added to the project had to be either a PNG or JPG and were displayed as another small widget called a file icon. These file icons allowed the user to see a preview of the image, the image's name, and allow the user to interact with files within the project. By clicking on a file icon the image, it represents would be displayed into the much larger image viewer and information about the file would be displayed in the information panel. Additionally, file icons could be dragged into the slide reel to add that image to the slideshow.

The slide reel was simple in design. It essentially housed a row of slide icons which are extensions of file icons with nearly the same functionality, just holding more information due to them references a slide instead of straight images. When the user drags a file or slide icon around the slide reel, they'll notice black bars appear between individual slides. When they drop what they are dragging over these black bars a slide will either be inserted or moved to that location in the slideshow.

The information panel was the longest in terms of code length simply because of the number of fields it displayed. It was broken up into two tabular sections that the user could switch between: slide/file info and project info. The project section displayed information like the name of the slideshow, number of slides, settings for the slideshow, the playlist interface used to add/remove/rearrange the playlist, and the tag interface used to add/remove tags from the slideshow. The slide/file info section housed similar information. When displaying the information about a file it ends up being much shorter displaying only file name, path, and buttons to create a slide from the image or remove the image from the project. When displaying slide information, it displays the same information about the image, file name and path, but also displays information about the slideshow such as the type of transition to use, how long the transition takes, how long the slide should be shown. It also has buttons to remove a slide from the slideshow and to preview the transition.

The image viewer widget is the only widget that's used in both the creator and player portions of the project. In the creator portion its main functionality is to just display an image. It somewhat displays the transition functionality when you click the preview transition button in the info panel, but it's not as polished as it is in the player program. Specific information on how transitions work will be explained in Chapter 2: Playing the Slideshow.

These four sections were all implemented into the main window using a Tkinter paned window, which essentially allows the user to resize each section independently to their liking. The final product of the creator UI can be seen in Figure 1. When having people test the functionality of the program one thing I got told a bit was how they would've preferred a different layout. Since each section's functionality is covered by a widget rearranging those widgets was extremely easy and just involved reconfiguring how the paned windows were set up. Since it was so easy there is now an option located in the menu bar to swap the layout

between the default and one that effectively moves the information panel from the bottom left to the top right as seen in Figure 2.

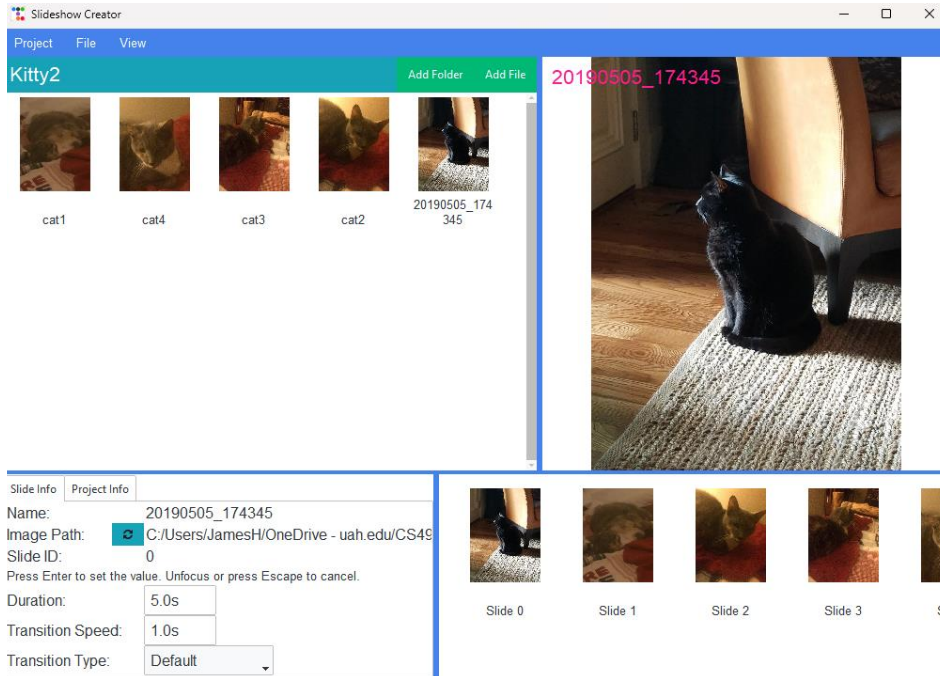


Figure 1: Slideshow Creator GUI Layout 1

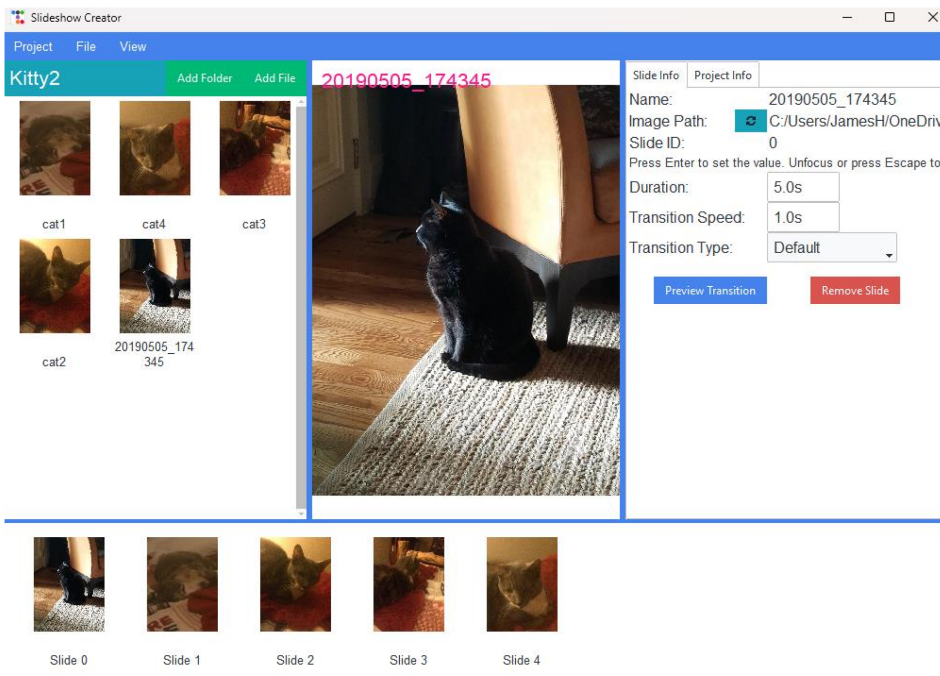


Figure 2: Slideshow Creator GUI Layout 2

Player

The UI for the player program was designed to be a lot simpler than the creator because its function is extremely simple. It is essentially the image viewer widget taking up the entire window with an overlay on top that has the controls. Controls include buttons to control the displayed slide itself such as previous slide, next slide, and pause. If a playlist exists for the slideshow, meaning the user has added music to the slideshow in the creator program, additional controls will be shown to allow control of music. These being next/previous song buttons, a progress bar to show song progress, and a timestamp to show how far along the song currently is. Pressing the pause button will pause the slideshow's automatic transitions, if the option is selected, and pause the music. Additionally, in the overlay there is a slide counter which tells the user what slide they are currently on. The overlay was set up in a way such that if the user interacted with the program through clicking or mouse movement it would appear and then disappear after a couple seconds of inactivity like a YouTube video. The appearance of the UI can be seen in Figure 3.

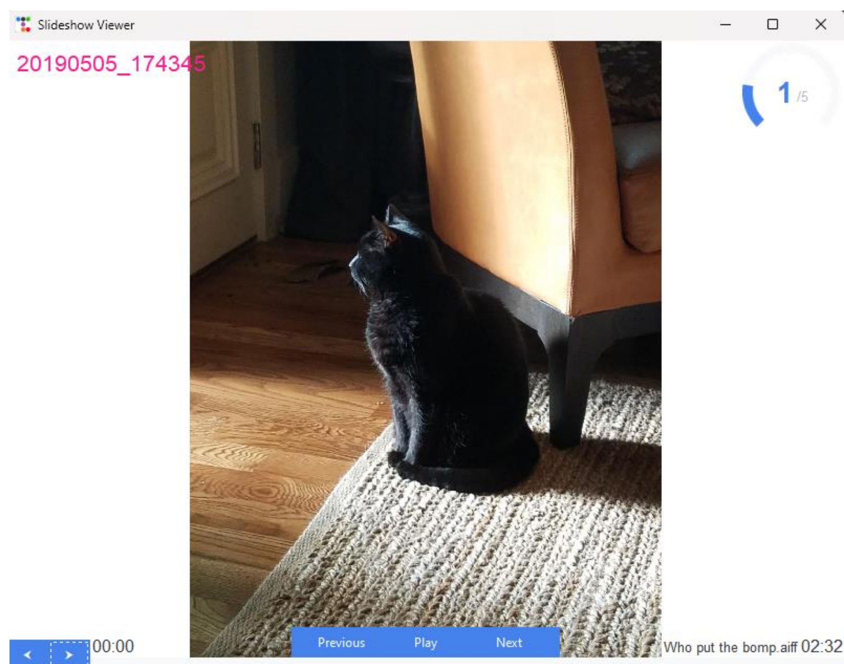


Figure 3: Slideshow Player UI Windowed

While the player is a lot simpler in terms of displayed UI than the creator, the logic to get there and implementing it made it just as much if not more work. For example, one thing the player program does when launched is it automatically puts the program in full screen. This is mainly done for image processing which will be explained in Chapter 2. Something I discovered while trying to implement the player UI is that full screening of a program in Tkinter is finicky. If you try to use the built-in functions it will only fill the screen of the main monitor, which was problematic because we wanted the ability to swap monitors the slideshow gets displayed on. To solve this, I had to full screen in an unusual way which had the side effect of removing the program icon from the task bar. The final solution was to create a dummy invisible window that would only be visible while the program was full screened and bind closing that dummy window to closing the player program. This way the player program is always in the task bar, and you could finally swap between monitors. The player program had a couple of fixes like these where a feature we expected to be simple turned out to be difficult. The process by which we did audio playing and the dynamic slide transitions will be covered in Chapter 2: Playing the Slideshow.

Start Menu and Other Functions

While the programs themselves are largely different they do share a common element which is their start menu. The start menu is set up such that on the right side there is a list of slideshows in the database. The list is in a tree view with a search bar and multiple columns you can search by. It is possible to search by name of the slideshow, the slideshow's ID, the tags attached to the slideshow, or the date it a slideshow was edited. By default, the list is ordered by the most recent slideshow. Clicking on a slideshow within this tree view list will highlight and select it. The top left of the menu displays basic information about the selected slideshow,

mainly the name, ID, and tags. The bottom left displays buttons that give the menu it's functionality. The appearance of the start menu can be seen in Figure 4.

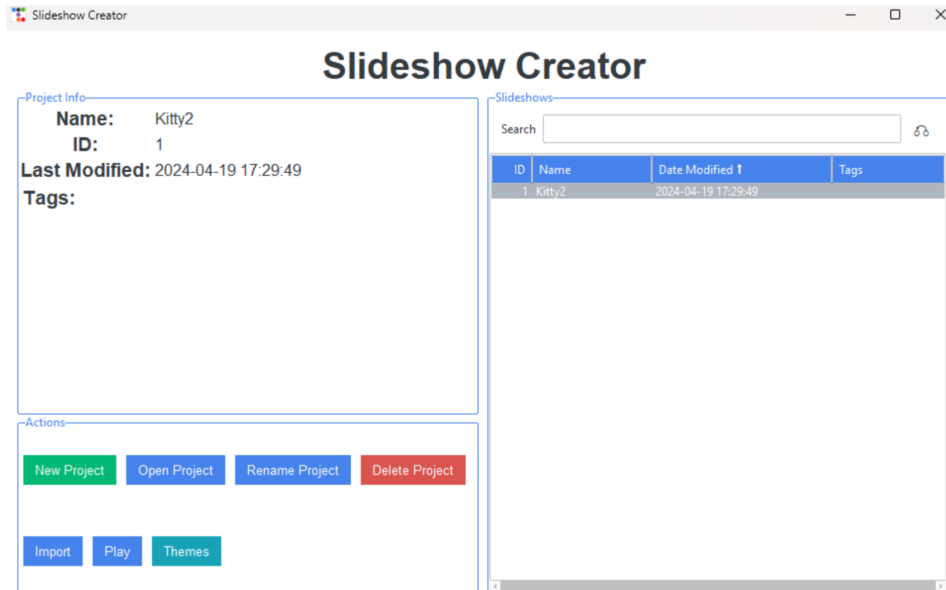


Figure 4: Slideshow Creator Start Menu

The buttons in the start menu mainly allow you to interact with the database. There are buttons to delete a project from the database, create a new slideshow to add to the database, import a slideshow from a file into the database, and play a slideshow in the player. The player start menu in particular lacks the new project button and the play button, instead just having an open and import button to load any slideshows. Additionally, there is a theme button which when opened displays a window where the user can select a new color theme. It is in the start menu that the user can select a slideshow project from the database and then load that project into the creator or player programs. More information on the database and how it works is in Chapter 3: Back End Design.

Chapter 2: Playing A Slideshow

The actual mechanics of playing the slideshow I felt were important enough to have its own chapter. A lot of the design choices we initially made when thinking about the slideshow player kind of fell apart once we started to implement them. In this chapter I will explain a bit of how the player program goes about processing the slideshow images, dynamically transitioning between slides, playing music, and many of the settings that can be set in the creator.

Most of the slideshow settings set in the creator relate to individual slides such as transition type, transition speed, duration of a slide, etc. However, there are a few settings that relate to the whole slideshow. Discounting adding songs to the playlist, these settings are slide shuffle, playlist shuffle, manual slide controls, and loop settings. Slide and playlist shuffle are self-explanatory. If they are enabled the order of the slides or songs are shuffled. Manual slide controls are a toggle on whether the slideshow changes the slides automatically or manually. The slideshow is automatic by default and slides will be displayed for however long a user set the duration for, but if this is enabled the duration gets ignored and it will only change slides when the user initiates it. The loop settings relate to how the slideshow loops. There are three options: indefinite, until playlist ends, or until slides end. Indefinite is the default and just means it loops forever until the user stops it. Until the playlist ends will play the slideshow until every song in the playlist has played at least once. If no song is present or playable then it just acts like indefinite. Until slides end will play until all slides have been shown. Instead of terminating the program when the slideshow ends it just pauses and stops the music.

Transitions

A lot of the core functions of the player program are handled by the image viewer widget discussed in Chapter 1. Effectively this widget is just a Tkinter canvas that you can load an

image onto. Just displaying the image was extremely easy. The hard part was implementing the transitions in a timely manner. Just doing the transitions wasn't that hard. The Python library Pillow allowed us to do all the necessary image operations and then display the final image in the viewer. However, we ran into problems with the timing. Our initial design was that we would use a method in Tkinter called after() that would call another method after a designated amount of time. The idea was that there would be a current slide and next slide. When the program makes the call to go to the next slide it would get information from the next slide such as transition speed, transition type, and the image file. It would then use the after method call to open the images, resize them, do the transition operations, and display the final image every 60-80ms i.e., 12-16 frames per second. This mostly worked, but the problem is that Python is slow, and these operations were expensive. Just opening an image and putting it on the canvas took about 30ms even on a good computer. The program could not do all the operations fast enough to complete the transition animation in the correct amount of time. To combat this, we had to change how we did the transitions.

Our new method was similar except it tried to reduce the number of operations necessary for the transition. We made it so when the project is first loaded into the player program it will pre-load all the images by opening and resizing them beforehand. These images were all stored in a list where their index was the position of their corresponding slide. This is the main reason the player starts out being full screened, we wanted all the images to be pre-rendered to be as big as they could be on their current monitor. This process happens again when you change monitors. Now instead of having to open an image file in every frame the program just does a deep copy from memory. This saved quite a bit of time on its own, but we also made it so instead of calling the after method every 80ms or so it would keep track of how long that specific frame took to render and then subtract that to get the remaining time. That way the frames all come at a

pretty constant rate assuming they finish rendering within 80ms, which most do. This basically solved our problem and fixed the timings, however if a particularly bad computer is used the transitions will probably not be timed correctly because displaying those transitions smoothly was prioritized over getting the exact timing right.

The general flow of transitioning between slides is that the player program will call for the next slide. It will then retrieve information about that slide such as transition speed, image files, and transition type. It will then send this information to the image viewer widget which begins a sort of transition loop that continues until the transition is complete. While doing this the program is constantly checking if the transition is completed, and once it does it loads the next image again just to be safe.

The program can do 6 different transitions: instant, fade, wipe up, wipe down, wipe left, and wipe right. Instant is just an instant change. Fade is just fading from slide to the next and involves blending alpha values. The wipe transitions all involve incrementally overlaying the next image on top of the previous image and a lot of cropping. The code for all transitions is similar which makes potentially implementing new ones quite easy. The only difficulty lies in finding the necessary Pillow operations.

Audio Playing

For audio playback we used the Python libraries Mutagen and Pygames. Mutagen allows the program to read audio files and get information such as the length of the song, sample rate, channels, etc. Pygames is what allows us to play the audio back to the user. It works by creating a background thread and playing the music there which allows for the GUI to remain functional. Other options were investigated such as using the libraries Pydub and Pyaudio but decided against using them due to the dependency on ffmpeg, a program that allows you to read/convert

audio files from one format to another. Technically speaking we could have tried packaging ffmpeg with the project itself, but we decided against that to keep the program light. Due to this the range of valid audio file formats is reduced by a large margin. Pygames also has the problem of being over kill just to play audio, but it fulfilled all our requirements, so we stuck with it.

The valid audio formats a user can add to the playlist are .wav, .mp3, and .aiff. Audio playback works by the player program creating an AudioPlayer object, loading a song into the object, and then playing/pausing the song through button presses. The program evaluates the slideshows playlist at the same time it preloads the images and when it finds a song it cannot load it just removes it from the playlist to be safe. If it is found no song can safely be played it just disables the audio player which removes the progress bar.

Chapter 3: Back End Design

Due to the project being very front end heavy, the back end ended up not being very large. It consists of two files: FileSupport.py and SQLSaver.py. SQLSaver.py deals with managing the slideshow database and database related functions. Meanwhile FileSupport.py has a bunch of important support functions and houses the class definitions that make up data structures like the slideshow, slide, playlist, and song.

FileSupport.py has a whole variety of miscellaneous functions. Some of them were straight up utility like a function to get the base name of a file path, or the file extension, or to format a timestamp. Two important groups really stood out though which were functions related to the cache and functions related to locating a file. A feature used regularly throughout the program is a sort of cache in the local app data directory. The program creates a folder named PySlideshow that has stuff like the user set color theme, the database file, and a cache folder. The cache folder is a place where images loaded into the creator/player are copied too as a sort of cautionary measure in case an image/song file goes missing. There are many functions in FileSupport.py that relate to initializing, validating, clearing, and saving files to this cache area. The other important functions in FileSupport.py are related to checking the location of a file. Basically, if a file is missing at its original location there are functions that will check if it was saved to the cache. If it was it will just use the cached version and tell the user, the original is missing. In the case of exporting/importing they will also check if the necessary files are in the same directory as the exported project file.

Data Structures

The program takes a pretty object orientated approach so many of the elements used in the program are designed as objects. FileSupport.py houses the class definitions for the slideshow, slide, playlist, and song objects. These objects are nested within themselves and have

a sort of hierarchy where the main object is the slideshow. Within the slideshow there exists a list of slides and the playlist, and within the playlist there exists a list of songs. It was designed this way because we felt each object was distinct enough to their own class. The only object type I feel was not strictly necessary is the playlist as its utility could just be rolled over into the greater slideshow object without much changing.

Apart from the slideshow object containing a playlist and list of slides, it also contains the slideshow settings such as manual controls, slide shuffle, loop settings. It also contains a list of files associated with the project, its database ID, its name, and a list of tags. Whenever a project is opened from the database all necessary information about the slideshow is retrieved and used to create the slideshow object. A slide object contains attributes such as its position in the slideshow, the file path to the image, the transition type, transition speed, duration it should be displayed, and the slides ID in the database.

The playlist object is simple and everything it can do could have been incorporated into the greater slideshow object. It is essentially just a list of songs and methods to control that list. There was a discussion made about whether we should remove it and fold it into the slideshow object, but we decided against it due to time constraints and that it wasn't strictly harmful to the code. The song objects are a bit more complicated. The attributes are simple as it's just the file path, name, duration, file type, and database ID. However, upon creation of a Song object it does do extensive checks that the song is of the correct file type (e.g., mp3, wav, aiff) and that the song can be loaded into Mutagen.

Database Design

The database is designed very similarly to the overall slideshow data structure and its children with some slight differences. It has a total of 7 tables: Slideshows, Slides, Songs, Tags,

TagRecords, Files, and FileRecords. The database lacks any sort of playlist table as it just rolled its settings into the slideshow table. The slideshow table also lacks any sort of reference to slides, songs, or tags. Instead, the slide and song tables contain a foreign key to their parent slideshow so to retrieve the slides/songs belonging to a specific slideshow one just searches the slide/song table for entries with a matching slideshow. Additionally, instead of referencing a file directly a join table is used to instead reference a file record. A file record basically keeps track of every instance a file is used. It mainly contains a reference to a file in the file table and an ID. Whenever other tables (e.g. slideshows, songs, slides) reference a file, it instead references a file record. This is to create a many-to-many relationship and the idea an individual file may exist in many slideshows, slides, or songs, but it's all one file. This saves us storage space because we aren't storing the exact same file multiple times and allows us to check if a stored file is even in use by checking to see if it has any records. The same principle is applied to tags where the tag table houses all existing tags, and a join table associates the tag and the slideshow. A diagram of the databases structure can be seen in Figure 5.

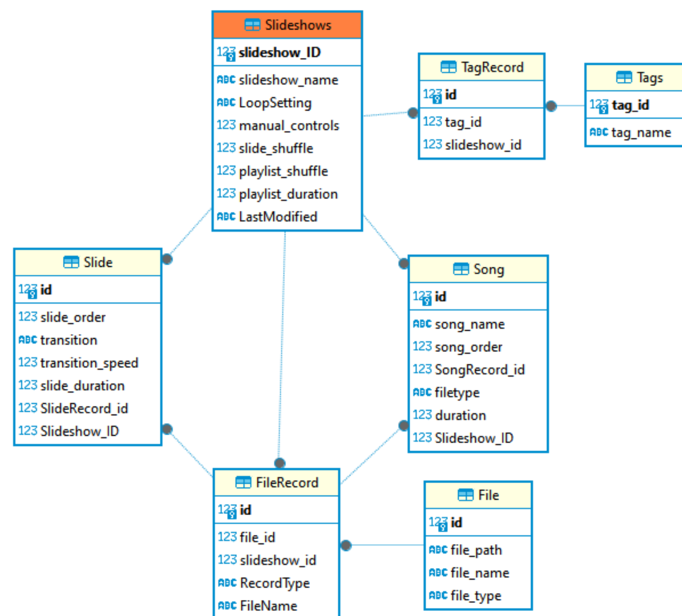


Figure 5: Database Diagram

For all database related operations, the program uses SQLSaver.py. It's mainly only used when the user starts the program, opens a project, or saves a project. Once the project gets loaded into the program, typically from the start menu, the database isn't really accessed anymore as all relevant data is just stored in the class data structures. The only time it really gets accessed afterwards is when the user initiates a save request in which that slideshow object is sort of parsed into the database.

File Formatting: JSON file vs Database

The main addition I made to the program to make it a capstone is that the original program my group and I made did not make use of a database at all. Instead, it made use of Pythons inbuilt capability to write object classes to a JSON file. Essentially saving a slideshow would just take the slideshow and its nested objects and dump it into its own JSON file. Then when the program loads a saved project Python can very easily just dump the JSON data back into the correct objects because it's all in the same structure. With the addition of the database this functionality is greatly reduced as all a user's slideshow data is just stored inside the database file so saving too or loading from a file is just not used as often. The only time the program makes use of it is when it exports or imports a project file.

There are pros and cons to each system. Just using the JSON files is a lot simpler and offers basically the exact same functionality. There aren't really many things you can do in the database version that you couldn't do in the JSON version. It's a bit more restrictive and insecure as with the database version the slideshow name must be unique and if something happens to the database all your projects are toast. However, the database version has a much larger capacity to expand in the future. For example, if I wanted to expand this software into a sort of cloud service

or app where users upload images/songs and create slideshows from them the database is a lot more useful to accomplish that. Instead of storing the database on the person's machine locally I could store it and the images on my own servers. I would need to add individual account functionality to the program and modify the database to assign slideshows to specific users, but a user would be able to access their projects from anywhere if they are connected online. In short, just using single JSON files for each project is simpler, more secure, and likely better for this specific project, but a database offers many more potential features one could add to their program.

Conclusion

Looking back after finishing the slideshow software there are many areas I feel like I could have improved. For example, one thing I noticed when making and using the slideshow creator program is that due to the number of elements and how they were implemented whenever the window gets resized it lags quite a bit. I wrote most of the code for this entire project and when I first started to implement the GUI, I had no experience with Tkinter and little GUI experience all together. Now after having created a program like this, I feel like if I had to do it again I could do it much better. The biggest improvement would be restructuring how I did the GUI as much of it near the start is kind of weird/unsafe and could be miles better.

This entire project has been a valuable experience to me as it required me to get a working knowledge of several programming skills. Through this project I have gotten to experience GUI programming, database design and implementation, event driven code, effective collaboration, and much more. I think this project effectively illustrates the culmination of my time here at UAH and how far I have come during my studies. This has been my capstone project; I hope you enjoyed it.