iExtract iOS Album to Folder Conversion Tool

Note: The "Team Process Description" section is located on pages 9 – 14.

Team Information:

- **(Names: Roles):**
    - Sam Daughtry: Developer
    - Noah Gregie: UI/UX
    - Kevin Gustafson: Director
    - Brendon Wong: QA Tester
- **GitHub Repo Link:** https://github.com/Gustakev/cs362-class-project
- **Communication Rules:**
    - We will communicate primarily through a Microsoft Teams group chat and our OSU associated emails. We may establish other means of communication amongst ourselves, but that is optional.
    - In terms of timeliness, as long as someone responds to a message sent via Teams or email within a few hours, it will be considered acceptable. However, there may be some times when quicker communication is expected and/or necessary to reach deadlines on time. During those times, it is preferable that team members respond to messages from other members within at least an hour, so that we may accomplish our goals on time.

Product Description:

- **Abstract:**
    - iExtract is a tool that helps people intelligently extract their photos and videos from their iPhones via reconstructing a folder structure and converting proprietary formats to common formats. This utility is dedicated to making iOS Photos app albums and collections easily convertible into folders en masse, which is not currently possible via any other tool than iMazing and a few other niche, monolithic tools. Additionally, iMazing has an expensive licensing model, as it provides many more features than just photo/video album to folder conversion, yet in a way that is not as focused.

- **Goal:**
    - The goal of this application is to solve the user experience issue regarding the either tedious, expensive, or iCloud-centric method of photo album and collection to local storage transfer that is available to iOS users at the moment. This app will facilitate a completely local transfer of albums and collections to local storage (converted into a folder-based format) on a user's PC, given they have created an unencrypted backup of their device, including their camera roll items.
- **Current Practice (Local Transfer Options):**
    - **Windows Transfer Tools:** Use the Windows Photos app (or older importer via File Explorer) to import your photos and videos without album/collection structure. Live photos get flattened via the Photos app. The older importer tool has the ability to organize photos and videos based on time, and it does preserve live photos, but the key photo and video of each live photo will not necessarily be kept together.
    - **iMazing:** The camera roll related features of iMazing closest resemble what iExtract is aiming to accomplish, but the issue with iMazing is that since it includes so many more features in addition to those ones, it has a high price tag and is subscription-based. This business model is not ideal for users that simply wish to convert their albums and collections into folders without having to manually reconstruct them from a raw dump, which is the kind of import method that the Windows Photos app provides for free.
    - **iTunes (Manually Copying User Created Folders of Photos/Videos from the iOS Files App):** This is a free method that anyone can use. However, the issues with this method are that it is entirely manual, tedious, wastes storage space on your phone during the process, and is prone to conversion failures, especially regarding live photos, which get converted into still images if not manually converted to videos before exporting to the Files App. Additionally, the album/collection structure must be entirely manually recreated by the user in order to preserve it, which is extremely tedious and infeasible for large numbers of albums, especially since in order to copy an album to your Windows computer, for example, you must first create a folder of the same

name in the iOS Files App, then you must ensure you convert all live photos to videos and re-add them to the original album, and then you must copy all of the items from said album into the folder of the same name in the Files App. After that, you must use iTunes to access the folder in which you stored these items and copy its contents to your computer.

- **Novelty:**
  - This application provides a cheaper and more focused method to users who want to avoid iCloud and backup their important albums and collections en masse, locally, in an automated manner. This application, unlike the alternatives, will not require an account, will be entirely free (or very cheap compared to the $29.99 per device price of iMazing), and will have power-user features, such as album blacklisting and whitelisting, ensuring users only back up the photos they want. It will be functional cross-platform, as it will use Python, which will allow it to be used on Windows, Linux, and macOS, or anywhere that has a Python interpreter for the version we decide to use. Additionally, unlike some tools (iTunes) which only allow users to back up their items via a tedious process if their phone is still functional, this application will work on unencrypted/decrypted iOS backups, not active phones, which allows for much more flexibility in terms of use-cases.
- **Effects:**
  - If we are successful in creating this app in the given timeframe, there will finally be a free (or very inexpensive) tool that may be used by anyone to locally backup their iOS devices' photos and videos without the friction that currently exists, such as cost and tedium.
- **Use Cases (Functional Requirements):**
  - User exports a single album:
    - Actors: Any iphone user
    - Triggers: The user wants to export an album to a folder of the same name.
    - Preconditions: The user must have an album already saved in their iPhone backup that they want to convert into a folder.
    - Postconditions (success scenario): The selected album has been successfully exported to the folder of the same name.

- List of steps (success scenario):
    1. User opens the CLI app
    2. User enters the proper command necessary to initialize the album export from the backup by using the whitelist feature to select it and export it, as a folder of the same name, to a selected parent folder
    3. The files in the album are successfully located (converted if needed or selected) and exported into the selected folder
    4. Program displays a success message indicating that the same number of files as the user selected have been exported
- Extensions/variations of the success scenario
    1. User enters the full command following the name of the application executable, e.g., "./iExtract --backup_location = ~/myBackup/, --whitelist = dogs, --dest = ~/extractedAlbums/, --convert = heic/jpg, mov/mp4, live_photo/mp4, --cq_img = 0.8, --cq_vid = 0.75, cq_live = 0.9"
        - cq is short for "conversion quality"
        - Note: Real flag system may work differently
    2. The files in the album are successfully located (converted in the specified qualities, since selected) and exported into the selected folder
    3. Program displays a success message indicating that the same number of files as the user selected have been exported
- Exceptions: failure conditions and scenarios
    1. When whitelisting an album to export, the user enters the name of a non-existent folder:
        - The program discovers that no such album exists within the specified backup. An error message is printed stating that there is no such album and the program returns to the main menu.
    2. When exporting the album, the parent folder for the new folder to be stored within, the destination folder, does not exist:

        - The program creates the non-existent folder and tells the user that the folder did not exist, so the program created it.
    3. When exporting the album, the user enters an invalid backup folder location:
        - As the program cannot extract anything, given it doesn't know where the backup folder is, it simply fails immediately, tells the user that they have offered an invalid path to a backup, and returns to the main menu.

- User exports albums containing overlapping items
    - Actors: A user with a highly organized album structure where multiple photos exist in multiple albums.
    - Triggers: The user wants to preserve their albums but does not want multiple copies taking up data.
    - Preconditions: The file system of the OS supporters shortcuts.
    - Postconditions: The "dog" folder contains the actual image, while the "animals" folder contains a shortcut to that file.
    - List of Steps:
        1. User initializes a full export of all albums.
        2. The system identifies that a certain photo is linked to both album A and album B.
        3. The album exports the real file into either A or B depending on which is first.
        4. When processing the second album, the system detects that the file has already been exported.
        5. Instead of copying the file, the system creates a shortcut pointing to the original file.
    - Exceptions: If the OS does not support shortcuts, then the system copies the full files everywhere they belong, and then it alerts the user that the shortcuts were not created successfully and, instead, the files were duplicated.
- Full camera roll backup
    - Actors: iPhone users
    - Triggers: The user wants to back up all their photos and videos.

- Preconditions: The user must have enough free storage on the destination drive partition that they're trying to save to.
- Postconditions: Files are stored without losing quality, and album/folder structure is preserved in the destination folder.
- List of Steps:
  1. User opens the app
  2. User selects "Back up the entire camera roll"
  3. App scans the photo library within the source backup
  4. App prompts the user to choose backup destination
     - Connected drive, network drive, cloud location
  5. User selects a destination
  6. App shows a summary screen:
     - Number of photos/albums to back up
     - Chosen destination path
  7. User selects "Start Backup"
  8. App shows a completion screen when finished

- Convert proprietary formats to common formats
  - Actors: User
  - Triggers: The user needs to view or share proprietary file types on a platform that only supports common formats
  - Preconditions: The media files have already been identified and iExtract has access to the local storage
  - Postconditions: All selected proprietary files are converted to standard formats
  - List of Steps
    1. User selects a folder of extracted files within the iExtract interface
    2. System scans for proprietary formats
    3. User selects the "convert to common format" option
    4. System converts all selected files into a standard format
    5. System notifies the user that the conversion was successful
  - Extensions/Variations: System either archives or replaces files based on the users preference.
  - Exceptions: If the file is unreadable, the system will mark it with an error tag before moving on to the next file.

- **Non-functional Requirements:**
  - Portability:
    - iExtract must work without a separate codebase for each platform on any machine that:
      1. Meets the system requirements, in terms of hardware.
      2. Has the ability to run programs of the Python version we choose to use.
  - Usability:
    - The error messages produced by the CLI must be clear and understandable to non-technical users.
    - The CLI must display a summary of the exported contents once an operation is complete.
  - Maintainability:
    - The architecture of the app will be modular such that different features are part of distinctly separate components, enabling features to be developed and/or fixed quickly without the risk of breaking other features.
- **External Requirements:**
  - Understanding the iPhone backup format:
    - This application can only function successfully if it is able to successfully parse an iPhone backup, which is external and unalterable to the developers of this app. Thus, the requirement that we study the backup structure of iPhone backups is tantamount to the success of this project.
  - Understanding the user's filesystem:
    - Operating system rules across Windows, Linux, and macOS must be followed in order to enable backups to be found and parsed into extracted folders.
  - Users expectations:
    - Users expect instructions and error messages to be clearly written and resolvable without our direct support.
  - The project must be open-source and buildable:
    - The source code for this application must be publicly available for others to download, enabling them to run it directly, without having to download a package or installer.
  - Consistent offline approach:

        - The tool must work offline, as it is made to perform pure offline iPhone backup extractions.

- **Technical Approach:**
    - We are planning to use Python and libraries that work across Windows, Linux, and macOS in order to create this app in such a way that it is functional across all the major desktop platforms. The main user experience will be a command line interface, but in order to reduce friction for less technical users, we will include a detailed list of commands that will be shown on-screen once the user enters "help". Additionally, we will be utilizing SQLite in our program to query the databases iOS uses to track which albums and collections media belongs to.
- **Timeline:**
    - **Week 3:** Ensure the architecture of the app, including its design, in terms of how the directories and modularity will be implemented, is finished, and the CLI minimally works without errors (no major features must be implemented at this point). The only important feature that should be finished by the end of this week is the ability to choose an unencrypted backup, use Manifest.db to locate Photos.sqlite, and then use the two databases in tandem to list every user-created album and every collection within the database.
    - **Week 4:** Ensure that the CLI is usable to bulk export every piece of media from the Photos app section of an unencrypted iPhone backup either via automatic flag mode or guided mode (which users will enter by simply starting the program with no flags).
    - **Week 5:** Ensure that the CLI blacklist and whitelist (which are internally the same thing, just with the blacklist logically inverted into a whitelist) work to specify which albums (or collections) you do and don't want to include in the extracted album (or collection) to folder conversions.
    - **Week 6:** Ensure that photos and videos present in multiple albums and/or collections at the same time are handled in the desired way by creating a folder of non-exclusive items and making a shortcut to each of those items and placing each shortcut into the correct folders, rather than wasting space copying the same files into multiple places. However, if the OS does not support shortcuts, or it does not have an analogous symbolic linking feature, which

shouldn't be the case, the fallback for the program will simply duplicate the items that aren't mutually exclusive to one album/collection into every single folder the shortcuts would have been present in.
    - **Week 7:** Ensure that conversion from Apple's proprietary image format can be converted into common formats such as JPEG or PNG at a specified quality during extraction.
    - **Week 8:** Ensure that conversion from Apple's proprietary video format (and from live photos) into common formats such as MP4 is possible at a specified quality level.
    - **Week 9:** Ensure that each and every known failure mode has a detailed error message associated with it that explains to a common user what they must do to mitigate the error in the future. Test the major features for bugs and usability issues and prepare for Week 10.
    - **Week 10:** Implement custom folder creation from albums/collections via advanced SQL query options choosable from the CLI.
    - **Week 11:** Bug test and polish the program.

Team Process Description:

- **Software Toolset:**
    - **Python:** We are planning on using Python to develop quickly and incrementally. Additionally, using this language will allow us to deploy the app on multiple operating systems using one codebase if we don't utilize platform specific libraries (or if we implement multiple platform specific libraries to facilitate the same actions on the 3 OSes we are planning on targeting: Windows 10+, Linux, and macOS.
    - **SQLite:** SQLite will need to be used for this application to function properly, as iPhone backups store a lot of data and metadata within SQLite databases such as Manifest.db and Photos.sqlite, which need to be parsed in order to reconstruct album and collection structure accurately in terms of folders. Moreover, Photos.sqlite contains metadata about images and videos, which we plan on preserving.
- **Team Member Roles & Justifications:**

    - **Kevin (Director):** This project needs a director so that the direction of the project stays on track and within scope. This role will prevent us from wasting time or not knowing what to do while waiting for others to finish their work. Kevin is being chosen for this role because the project was his idea, so it makes sense that the general direction of it should stay in line with that idea.
    - **Noah (UI/UX):** A good program requires intuitive controls and a navigable menu. We want our program to be user-friendly and easy to use. Noah is being chosen for this role because he has prior experience in creating UI and designing websites.
    - **Brendon (QA Tester):** Having a QA tester ensures that we catch functional failures and edge-cases. The person taking on this role will need to create test cases and give us criteria for correctness. He will also spend time to validate that the product is usable and reliable for users.
    - **Sam (Developer):** This program requires developers to ensure that the project is functional and meets the product description. This role will work on the back end of the program to build up all the required features and will be continuously reviewing the code to ensure that the quality is upheld throughout the codebase. Sam is being chosen for this role because he has prior experience in working as a back end developer in previous projects.
- **Schedule:**
    - **Week 3:**
        - Director: Jumpstart development on the program's ability to list albums and collections from an iPhone backup. Host team meeting to confirm skeleton code is working to reach the goal for week 3 stated in the Timeline section. Update the living document.
        - UX/UI: Create the initial design idea (beyond the minimally functional CLI). Identify user flow path.
        - Developer: Create skeleton code for all features, ensuring modularity and structure are baked into the design of iExtract. Create mock data, and start working on feature 1.
        - QA: Review all use cases and draft test cases for features 1 and 2. Prep the regression test sheet.
    - **Week 4:**

    - Director: Host the team meeting, help develop features 1 and 2, and update the living document and weekly report.
    - UX/UI: Walk team through user flow + transition. Adjust flow based on feedback. Finalize UI design for features 1 and 2.
    - Developer: Start working on feature 2 and push first design so QA can test.
    - QA: Draft test cases for features 3 and 4. Add a UI validation scenario. Update the regression test sheet.
  - **Week 5:**
    - Director: Host the team meeting. Update the living document and weekly report. Coordinate with QA testing to make sure tests run smoothly. Revise features 1 and 2 if needed.
    - UX/UI: Update the layout, add consistent styling rules, and finalize features 3 and 4.
    - Developer: Revise any issues with features 1 and 2, start working on features 3 and 4, and push the initial design.
    - QA: Run test case for features 1 and 2. Update test cases after discussing results with the team.
  - **Week 6:**
    - Director: Update the living document and weekly report, run a progress check meeting, and help revise features 3 and 4.
    - UX/UI: Pair with QA to test flow + identify usability issues, and then update the design.
    - Developer: Revise any issues with features 3 and 4
    - QA: Run test case for features 3 and 4 and add edge case test cases to test.
  - **Week 7:**
    - Director: Host weekly meeting, update the weekly report and living document, finalize features 1 and 2 and coordinate with QA to make sure there are no bugs left to fix.
    - UX/UI: Fix any UI errors regarding navigation or unclear error message concepts.
    - Developer: Add validation + error messages and fix any errors.
    - QA: Test edge cases and retest any previous issues.
  - **Week 8:**

        - Director: Host weekly meeting, update the weekly report and living document, and coordinate with the QA to make sure all tests are running fine. Help revise any edge cases.
        - UX/UI: Retest and make sure the UI runs with the backend.
        - Developer: Help integrate the UX/UI design to work with the backend seamlessly.
        - QA: Execute a full test with all use cases. Check UI flow and usability.
    - **Week 9:**
        - Director: Call the final weekly meeting, update the weekly report and living document, and run final tests on the app.
        - UX/UI: Finalize the design and run a final test.
        - Developer: Test the final draft of the app for stability.
        - QA: Run verification on the final build and validate acceptance criteria for each test case.

- **Major Risks:**
    - **Scope Creep:** If the planned features are too large in scope for us to complete within the given timeframe of the class, we won't be able to complete the project in the way that we originally desired. To avoid this issue, we will develop incrementally, adding basic features first, which will compose a minimum viable product before we move on to adding things like advanced filtering and detailed metadata preservation.
    - **Team Members Not Communicating:** If people don't communicate with each other accurately or timely, then it will be very difficult to work on the project, not knowing what other people have done or are planning on doing. This could cause the team to reach a bit of a stalemate, in terms of development speed. To avoid this, we will all decide on which features each of us will be working on each week in advance. This strategy will allow us to avoid working on overlapping or conflicting features, and it will allow us to know which responsibilities are our own.
    - **Difficulty of Understanding the Backups:** One of the most significant risks to the success of this project is that it will be too difficult to understand the format in which Apple stores and organizes camera roll files, in regard to the metadata and

album/collection structure that we wish to preserve using the folder-based backup system. To mitigate this risk, we will do research into the existing documentation around how iOS stores camera roll files and their metadata, which is publicly available but perhaps difficult to understand.

- **External Feedback:**
    - **Export Everything Feature Complete:**
        - At this point, we will need to ensure that the export actually exports everything, not including thumbnails and other files that are meant to be temporary and aren't shown in the Photos app on iOS. To do so, we may recruit some people we know in order to test this tool on backups of their phones, as well as to test this feature in front of TAs and/or the professor to ensure that the code is not only functional, but that it is also efficient and complies with common standards.
    - **Export Only Certain Albums/Collections Feature:**
        - At this point, we will need to ensure that when a user uses the application with the intention of exporting only certain albums and collections, only those are included in the export, and that all the real items are included in each exported album and collection, without the temporary files and thumbnails. Once again, to do this, we will need to have multiple people, some that we know in our personal lives, and perhaps some TAs, attempt to use the tool for this purpose and demonstrate its usage on backups of their devices. We will also need to have TAs and other people with expertise look over our code and help us catch bad practices early on, so that we don't build up technical debt.
    - **Conversion Quality Testing:**
        - When a user chooses that they would like to convert proprietary file types like live photos into common file types, those conversions need to be done properly in every case. In order to ensure that this is true, we need to test the program on multiple backups from different devices, ensuring consistent behavior. We will likely need external sources to volunteer backups for us to perform operations on, or we will need to create more backups to examine ourselves.

Alternatively, we could have people try out the tool on their own time, so that we don't need to invade their privacy, and they could tell us if the conversion process was successful or not, allowing us to get real world "customer" feedback.

Major Features (4+):

1. Mass Album -> Folder Conversion
2. Mass Collection -> Folder Conversion
3. Folder & Collection Blacklisting (And Whitelisting) System
   a. This feature will enable users to choose which albums and collections they want to convert into folders via command line flags.
4. Proprietary File Type -> Common File Type Conversion Options

Stretch Goals (2+):

1. Extract Folders of Files from iOS Files App & Incorporate Blacklist (And Whitelist) Features
2. Custom Folder Creation Implemented Via SQL Filtering
   - The user would use a UI to apply the filters, not needing to enter the commands themselves.
3. Implement a GUI Alternative to the CLI
4. Add AI Descriptions of Content to Photo Metadata

Notes:

- The major features should constitute a minimal viable product (MVP).