Issue Resolution Report

Group 14
DD2480 Software Engineering Fundamentals
KTH Royal Institute of Technology
Jesper Lindeberg, Hugo Sacilotto, Hanzhi Zhang, Jeffrey Chang, Yuning Wang

March 7, 2025

1 Project Overview

Name: Immich-Go

Repo URL: `https://github.com/simulot/immich-go`

Fork URL: `https://github.com/DD2480-Group-14-2025/Assignment4-immich-go`

1.1 Purpose

Immich-go is an open-source command-line utility designed to facilitate bulk upload operations to an Immich server, a self-hosted media management solution optimized for efficient photo and video organization. The tool provides a lightweight, scriptable alternative to the Immich web and mobile clients, making it ideal for users who need to manage large-scale media transfers.

The primary objectives of Immich-go include:

- Enabling users to upload and synchronize media files with an Immich instance.
- Supporting efficient file filtering and selection through type-based inclusion/exclusion.
- Handling large datasets with minimal overhead, making use of concurrency where necessary.
- Providing a robust CLI interface for automation and integration with existing workflows.

1.2 Architecture

Immich-go follows a modular architecture that interacts with an Immich server using its API. The core components of the system include:

- File Handling and Filtering
  - Identifies media files from the specified directories.
  - Applies inclusion/exclusion rules based on file types such as images, videos, sidecar files, and useless files.
  - Ensures sidecar files are processed alongside their main media files.
- Concurrent Upload Manager
  - Manages parallel uploads to optimize performance.
  - Ensures efficient error handling and retry mechanisms.
  - Uses API authentication to establish a secure connection to the Immich server.
- Metadata Extraction and Processing
  - Extracts and includes metadata such as timestamps and geolocation when available.
  - Supports file deduplication and integrity checks before uploading.
- CLI Interface
  - Offers various command-line options to configure the upload process.
  - Supports flags like –include-type and –exclude-type to filter files.

A high-level representation of Immich-go's architecture is as follows:

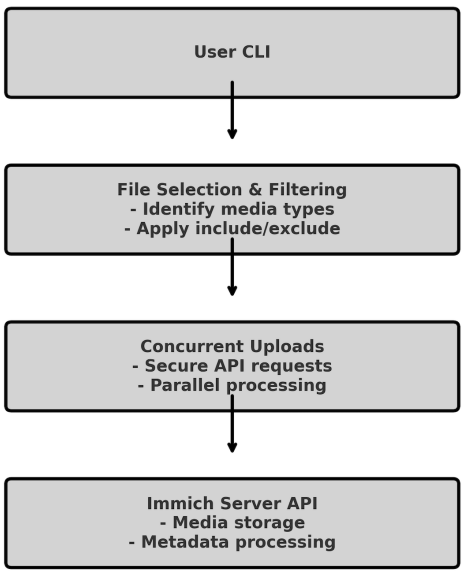

Figure 1: A high-level representation of Immich-go's architecture

2 Onboarding Experience

Did you choose a new project or continue on the previous one? If you changed the project, how did your experience differ from before?

For this assignment the decision was made to switch to a new project, *immich-go*. The onboarding experience was a lot more hands on than our previous project and was less documented without any contributing guideline. However, this new project had fewer abstractions and more readable code compared to the previous one, *apache commons-imaging*.

It was simple to build and test the project as the only required dependency was the Go programming language. It was however a bit more time consuming to set up a local immich server, but it could be done within 30 minutes. The README included good usage examples which made it easy to start experimenting with immich-go. What was missing was any kind of contribution guidelines, though it was easy to get in contact with the maintainer of the project who was very helpful.

3 Effort Spent

Jesper Lindeberg - 21 $\pm$1 hours

- Plenary discussions/meetings: 1 hour
- Discussions within parts of the group: 4 hours
- Reading documentation & code: 5 hours (no onboarding for the repo)
- Reading documentation & code (Cobra & flag package): 2 hours
- Configuration and setup: 3 hours
  - Docker 2 hours (download + configure)
  - WSL setup 30 minutes (debug + mem issues with docker)
- Analyzing code/output: 2 hour (pr reviews and debug tests)
- Writing code: 3 hours (had to rewrite tests due to function change)
- Running code: 20 minutes

Jeffrey Chang $\sim$ 20 hours

- Plenary discussions/meetings: 1 hour
- Discussions within parts of the group: 2 hours
- Getting a basic understanding of Go: 4 hours
- Getting a basic understanding of PlantUML: 2 hours
- Writing the code for the UML: 4 hours

- Reading documentation & code: 4 hours
- Writing parts of the report: 1 hour
- Configuration and setup: 2 hours

Hugo Sacilotto ~ 21 hours

- Plenary discussions/meetings: 1 hour
- Discussions within parts of the group: 4 hours
- Installing Go: 30 minutes
- Reading code in the repo: 3 hours
- Reading documentation (immich-go repo, Go, cobra): 3 hours
- Discussions with the maintainer of immich-go: 1 hour
- Setting up a local immich server: 30 minutes
- Testing our code on local immich server: 2 hours
- Analyzing and reviewing code: 3 hours
- Writing and refactoring code: 3 hours

Hanzhi Zhang - 20 hours

- Plenary discussions/meetings: 1 hour
- Discussions within parts of the group: 3 hours
- Learning the basic knowledge of Go: 4 hours
- Reading through code and documents of the repo: 5 hours
- Programming the and modifying the code related to –include-type command flag: 4 hours
- Set up a local immich server and testing the code on it: 2 hours
- Reviewing codes: 1 hours

Yuning Wang - 23 hours

- Plenary discussions/meetings: 1 hour
- Discussions within parts of the group: 3 hours
- Acquiring a basic grasp of Go: 5 hours
- Reading through codes and documentations of the repo: 6 hours
- Analysing the issue and documenting the requirements: 2 hours
- Reviewing codes: 2 hours

- Contributing to the report: 4 hours

4 Overview of Issue(s) and Work Done

Title: Feature Request: Add Option to Upload Based on File Type (Image or Video) Without Specifying Extensions

Issue #739

URL: `https://github.com/simulot/immich-go/issues/739`

Summary: The issue is a feature request to be able to add a flag in the command to upload files to specify what file type should be included (image, video, etc). Currently, file extensions can be specified that should be included or excluded but there is no functionality to simply choose the file type that should be uploaded. This can be a useful feature for users that, for example, have folders with lots of different files and many different file extensions for image or video files.

Scope: Describe the affected functionality and code areas.

5 Requirements for the Feature or Refactored Functionality

Below are the documented requirements for the feature implementation.

5.1 REQ-001: Media Type Filtering via `--include-type` Flag

Description: The system must allow users to specify a media type (`IMAGE` or `VIDEO`) rather than manually listing file extensions.

- When `--include-type=IMAGE` is set, all supported image file extensions should be included automatically.
- When `--include-type=VIDEO` is set, all supported video file extensions should be included automatically.
- The inputs are case-insensitive.

5.2 REQ-002: Combined Filtering with `--include-extensions` and `--exclude-extensions`

Description: Users must be able to combine `--include-type` with the existing `--include-extensions` and `--exclude-extensions` flags.

- Example:

```
immich-go upload from-folder --include-type=IMAGE \
  --include-extensions=.mp4 --exclude-extensions=.jpg
```

- Includes all images, MP4 files, but excludes JPG files.
- The system should correctly merge these flags without conflicts.

5.3 REQ-003: Default Behavior When No Flags Are Provided

Description: If no filtering flags (`--include-type`, `--include-extensions`, `--exclude-extensions`) are provided, the system must default to including all supported media files and handle sidecar files properly.

5.4 ADD-REQ: Support Across Different Upload Options

Description: The `--include-type` flag must be available for all upload and archive commands, including:

1. `from-folder`
2. `from-google-photos`
3. `from-immich`

This is an additional requirement that we implemented based on the developer's review following the submission of our pull request to the main repository.

5.5 Tests Corresponding to Requirements

Table 1: Correspondence of tasks to the requirements

Requirement	Test Case	Source	Notes
REQ-001	`TestStringList_IncludeType`	New	Ensures correct file types (image/video) are included.
REQ-003	`TestStringList_IncludeType`	New	Tests that sidecar files are included along with images/videos.
REQ-001	`TestStringList_IncludeType_Set`	New	Ensures invalid file types are correctly rejected.
REQ-001	`TestStringList_IncludeType_Set`	New	Ensures valid file types are correctly accepted.
REQ-001	`TestStringList_IncludeType_Set`	New	Tests different cases (lower/upper/mixed) for type input.

6 Code Changes

6.1 Patch

Include the relevant code changes, or provide a Git command: The code changes can be seen by running the following command (from the main branch)

```
git diff c25529d
```

where `c25529d` is the id of the most recent commit in the main branch before we started working.

Optional:

- Corresponding to the 4th point for P+, the patch is checked to be clean (no commented-out obsolete code, no unnecessary whitespace changes).
- For the 5th point for P+, the patch has been submitted to the project and is pending acceptance. The PR can be reviewed here

6.2 In Context of Software Architecture and Design Patterns

- Adherence to the Single Responsibility Principle (SRP)

  To implement the new feature, structs and functions are modified or newly added, so as to undertake a cleared and focused responsibility respectively.

  - `IncludeType` restricts values to `IMAGE` or `VIDEO`, enforcing constraints.
  - `setIncludeTypeExtensions()` dynamically applies the correct file types based on user selection.
  - `MediaToExtensions()` encapsulates the media-type-to-extension mapping logic by a reverse lookup map.

  This ensures better separation of concerns, making modifications to media-type mappings localized and less error-prone.

- Alignment with the Open-Closed Principle (OCP)

  The `setIncludeTypeExtensions()` function automatically picks up any new file types added to `DefaultSupportedMedia`. By using `MediaToExtensions()`, the system can easily accommodate new media types (e.g., `AUDIO`, `DOCUMENT`) without modifying the core filtering logic. This design aligns with the Open-Closed Principle (OCP)—new functionality can be added without modifying existing code, reducing the risk of regressions.

- Encapsulation Principles and Factory Method

  The use of custom types like `IncludeType` encapsulates logic within their respective types instead of spreading it across multiple locations. The `IncludeType.Set()` method validates and normalizes user input, following the Factory Method pattern to ensure valid instances.

7 Test Cases

7.1 Test Results

Summarize the test results before and after changes. Provide links to logs if applicable.

Our issue was a feature request so there were no failing tests initially. We implemented some unit tests to functions that we wrote. All the tests are passing. The test functions (`TestStringList_IncludeType()` and `TestStringList_IncludeType_Set`) can be found in the file `internal/cliFlags/stringlist_test.go`. More information about the specific tests can be found in Table 1.

8 UML Class Diagram and Description

This issue highlights a need to filter uploads by the user based on the file's content type, i.e. `VIDEO` or `IMAGE` than just manually choosing all the file extensions and the solution involves adding new CLI flags and mapping mechanisms that can automatically select file extensions given a media type.

Figure 2 shows the affected classes of this change to solve the issue. By introducing the `--include-type` flag and the associated type `IncludeType`, the code now lets the user to filter uploads based on the file content rather than manually selecting all file extensions. The function `setIncludeTypeExtensions()` automatically populates the list of allowed file extensions by using the mapping provided by `MediaToExtensions()` in the `supported.go` module.

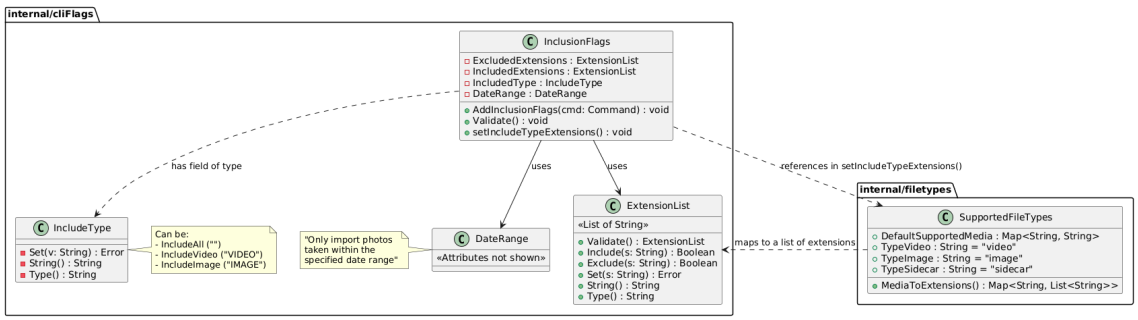

Figure 2: UML of the Affected Classes

9 Overall Experience & Reflection

9.1 Experience

The existing project documentation was partially helpful, but some aspects required deeper investigation. Guidance and examples are missing for some parts, which causes difficulties to start with the project.

The communication with the project's community is both delightful and helpful. Developer Simulot responds promptly and provides us with valuable suggestions.

9.2 Benefits

- Improved user experience by simplifying file-type selection.
- Enhanced code clarity with modular filtering logic.

9.3 Drawbacks & Limitations

- Complexity trade-off: Introducing a new filtering mechanism required careful handling to avoid conflicts with existing flags.

- Challenges in testing some requirements: Some of the requirements are hard to be covered by unit tests, while manual checks on local servers have been done to ensure the proper functionality.

9.4 Key Takeaways & Learnings

This project provided valuable insight into both the principles of software design and the challenges of practical implementation.

- Encapsulation & Design Patterns: We reinforced the Single Responsibility Principle (SRP) by structuring filtering logic into dedicated functions and ensuring modularity.
- Importance of organized documentations: It takes a lot of time to get used to a repository and reading up on the relevant code + documentation. A well written onboarding guide can greatly speed up this hurdle when contributing to a project.

9.5 Assessment across different SEMAT alphas

- Way of Working: It has been improved since the first assignment and so on, as we get to handle feedback from others effectively.
- Team: The collaboration within the team and within the developing community is effective.
- Opportunity: The `--include-type` flag was proposed in response to usability concerns, as users found it cumbersome to specify file extensions manually.
- Requirements: We followed a structured approach to defining requirements, ensuring traceability between implementation and validation.
- Software System: The feature integrated seamlessly without breaking existing functionalities.
  - Used existing file type mapping functions, ensuring consistency.
  - Kept backward compatibility with extension-based filtering.

10 Pass with Distinction

Below are notes on how we believe we have achieved the P+ requirements.

1. The architecture and purpose of the system are presented in **1 Project Overview**.
2. Updates in the source are put into context with the overall software architecture and discussed, relating them to design patterns and/or refactoring patterns in **6.2 In Context of Software Architecture and Design Patterns**.

3. Relevant test cases are traced to requirements in **5.5 Tests Corresponding to Requirements**.
4. Our patch is checked to be clean.
5. The patch has been submitted, and considered for acceptance. The Pull Request can be seen here `https://github.com/simulot/immich-go/pull/796`.
6. We have a critical reflection of our work in **9 Overall Experience & Reflection** .
7. We actively communicated with the project's developer and, based on their requests, expanded the issue's feature to a more general level, which is documented in **5.4 ADD-REQ: Support Across Different Upload Options**.