Report for assignment 4

Project

Name: TagStudio

URL: https://github.com/TagStudioDev/TagStudio
Fork: https://github.com/SoffanDD2480/assignment-4-ISSUE-RESOLUTION/blob/main/CONTRIBUTING.md

“TagStudio is a photo & file organization application with an underlying tag-based system that focuses on giving freedom and flexibility to the user. No proprietary programs or formats, no sea of sidecar files, and no complete upheaval of your filesystem structure.”

Onboarding experience

We chose to work on a different project for this assignment since the last one had a very active community and it was hard to find an issue that wasn’t already fixed, let alone find an issue that was complex enough to fit the qualifications for this assignment. Also to effectively address any issues not fixed yet, we would need to be thoroughly familiar with the entire codebase, which could be challenging given its size of over 20,000 lines.

For this assignment, we didn't encounter many issues during onboarding. There were some differences between Windows and Mac, but the only detail missing from the documentation was the need to have PyQt installed on our machines. Additionally, we were required to use Python 3.12.*, and the documentation recommended setting up a virtual environment. Running the command `python3.12 -m venv .venv` created the virtual environment with the right python version. Then activating the virtual environment and lastly installing all dependencies using the command `pip install requirements-dev.txt`.
The onboarding experience was very similar to the last project that we did for assignment 3.

Effort spent

For each team member, how much time was spent.

Felicia Murkes	**Hours spent**	**Comments**
plenary discussions/meetings	2	

discussions within parts of the group	2	
reading documentation	2	
configuration and setup	1	Had to spend some time figuring out that I should've used requirements-dev.txt instead of requirements.txt. Also had to download PyQt which caused some issues before realising i had to download it using brew on mac. I also had to solve the issue of sometimes getting a PyQt error when running the application
analyzing code/output	8	The documentation was really bad in the dose so we had to analyze a lot of the code to understand it
writing documentation	6	Working on the report
writing code	3	There were not that many lines that needed to be added
running code?	1	Test implementations and tests

Elias	**Hours spent**	**Comments**
plenary discussions/meetings	2	Discussion regarding choice of project and issue to work on
discussions within parts of the group	2	
reading documentation	30 min	
configuration and setup	10 min	
analyzing code/output	10	The code was not very well documented, so it took some time to analyze and understand what everything does and where we could implement the feature

writing documentation	1	
writing code	5	Studying another issue, help with 766 as well as test code.
running code?	2	Testing implementations.

Riccardo	**Hours spent**	**Comments**
plenary discussions/meetings	2	
discussions within parts of the group	2	
reading documentation	2	
configuration and setup	30 minutes	
analyzing code/output	7	Reading the code in order to understanding in which class we should had to work on
writing documentation	2	
writing code	3	Implemented tests for the implemented function
running code?	2	Testing implementation

Albin	**Hours spent**	**Comments**
plenary discussions/meetings	2	
discussions within parts of the group	2	
reading documentation	4	
configuration and setup	30 min	
debugging	8	During the project I encountered a persistent issue where the application would freeze after switching images once regardless of my environment as I managed to reproduce my

		issue in docker. This bug was not officially reported but it consistently occurred for me. After extensive debugging and analysis I managed to resolve the issue and document my findings in a new issue report. However the main development team at TagStudio has not been able to reproduce the bug on their end. Despite this resolving this problem took up most of my time but it did contribute to my understanding of the projects codebase.
writing documentation	4	
writing code	1	
running code	2	

Dmitry	**Hours spent**	**Comments**
plenary discussions/meetings	2	
discussions within parts of the group	1	
reading documentation	5	Focusing on the contribution process
configuration and setup	1	Had similar issues to Felicia, reinstalling the codebase and remaking a clean venv for the third time fixed the issues in the end, as well as installing some missing dependencies manually.
analyzing code/output	4	Getting familiar with the code base, how to write them in an acceptable format
researching/writing the report	3	SEMAT, UML diagram, adding to P+ parts

refactoring code	4	Refactored code to the guidelines of the original project, made an acceptable patch request as a PR

For setting up tools and libraries (step 4), enumerate all dependencies you took care of and where you spent your time, if that time exceeds 30 minutes.

There was a lot of time spent on trying to understand the code since the documentation was lacking a lot. There were not really any good descriptions of what each file did, we had to figure out a lot what the code does so a lot of time was spent on it.

Overview of issue(s) and work done.

Title: [Feature Request]: Search images by aspect ratio

URL: https://github.com/TagStudioDev/TagStudio/issues/766
Where we worked on the issue:
https://github.com/SoffanDD2480/assignment-4-ISSUE-RESOLUTION/tree/Issue/766-search-aspect-ratio

Summary in one or two sentences:

Define an aspect ratio range as a search term, for example something like:

- Search for square images:
  - aspect_ratio:1
- images taller than they are wide:
  - aspect_ratio:<1
- images wider than they are tall:
  - aspect_ratio:>1
- images that were in a range, say standard non-ultrawide monitor aspect ratio:
  - aspect_ratio:1.6-1.77

Scope (functionality and code affected).
This issue affected 5 source files

`ts_qt.py`
`ast.py`
`enums.py`
`models.py`
`visitors.py`

Added Functionality
We added a search parameter for aspect_ratio that allows users to filter images based on their aspect ratio. This parameter supports auto-completion in the search bar, making it easier for users to input valid search queries. The aspect ratio data is stored in the database, and the search functionality filters images based on this stored aspect ratio.

Implementation
To implement this functionality, we first ensured that the entries table in the database includes an aspect_ratio column. We updated the SQLAlchemy model to reflect this change. For each entry the aspect ratio is calculated and added to the column. In the update_completions_list method, we added logic to handle the aspect_ratio query type, providing auto-completion options. We also implemented the filter_by_aspect_ratio method to calculate and compare aspect ratios, filtering images accordingly. This method directly accesses the list of images stored in the Library class.

Requirements for the new feature or requirements affected by functionality being refactored

R1: Autocomplete
When the user starts typing “aspect_ratio” in the search bar, there should be an option that pops up below the search bar for the user to click on to make sure the right query is being made.

R2: Added column to database
The database that stores the information of each file should also contain an aspect ratio column to allow comparison to the search query/input.

R3: Search with ratio
The user should be able to search for images with a specific ratio, for example, 1 for square.

R4: Search with < or >
The user should be able to specify an aspect ratio threshold in which the query returns images with an aspect ratio less than or more than the threshold.

R5: Search with a range
The user should be able to search for images that have an aspect ratio within a range

Code changes

Use following command to show code changes

```
git diff Issue/766-search-aspect-ratio..main
```

Patch

Link to a PR in our organisation, in an acceptable format following the guidelines of the project:
https://github.com/SoffanDD2480/assignment-4-ISSUE-RESOLUTION/pull/1

The patch was cleaned to fulfil the “clean patch” requirement for P+ (point 4 in P+), which can be seen in the following commits:
https://github.com/SoffanDD2480/assignment-4-ISSUE-RESOLUTION/pull/1/commits/a6416d5dc02a4a933a3dfefaeed753f0f33b6f04

https://github.com/SoffanDD2480/assignment-4-ISSUE-RESOLUTION/pull/1/commits/0a94a2a7c077f4196663461e3bf4eea7710fcc6b

Automated checks

The code passes all automated tests / github actions.

Test results

Overall results with link to a copy or excerpt of the logs (before/after refactoring).

Tests before:
---------- snapshot report summary ---------- 6 snapshots passed. ========== 167 passed, 312 warnings in 15.48s ==========

After changing the code and fixing the issue, all tests in the file test_search.py were failing. This was because during our fix we had to change a table in the database which led to the test database missing a column, this was easily fixed and it needed to be fixed either way since we had to add tests for our new feature.

Added tests to test following aspects:

1. Test to search for aspect_ratio:1(or any fixed number)
2. Test to search for aspect_ratio with upper bound
3. Test to search for aspect_ratio with lower bound
4. Test to search for aspect ratio within a range
5. Test for zero returns in test_single_constraint
6. Tests for color + aspect ratio in test_and
7. Tests for filetype + aspect ratio in test_and
8. Test for color or aspect ratio in test_or

9. Tests for color + aspect ratio in test_not
10. Tests for aspect ratio in test_not
11. Test for test_parentheses

The test suite that was already given for the searching feature was very easy to add to. We just needed to add a line in the existing tests of what search query we wanted to test, and the “count” should be the amount of expected files after searching.

All the tests added in our case relate to requirements R3-R5 (point 3 in P+). The first and fifth tests are directly related to requirement R3, where it’s tested if the program is able to input different kinds of aspect ratios. Tests two and three are related to requirement R4, where both test the less than or more than operators. Test four tests if the range functionality of R4 is implemented. The rest of the tests are also related to those three requirements, where those are tested in other scenarios, in conjunction with the existing features, to see if the new feature can integrate with those.

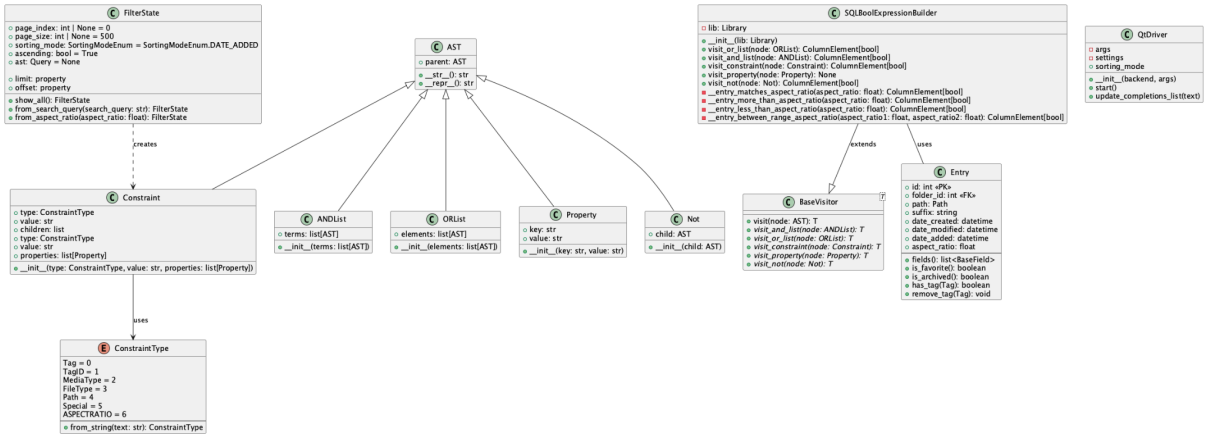
After adding 11 tests:
---------- snapshot report summary ----------
6 snapshots passed.
========== 178 passed, 344 warnings in 24.96s
==========

UML class diagram and its description

In this UML diagram we for the most part only included the functions that were effected. There are some functions/classes that were not effected but are there to easier explain what each effected class does.

The AST class serves as the foundation from which specialized components inherit. Logical operations are handled by ANDList (for conjunction), ORList (for disjunction), and Not (for negation) classes, allowing for more complex query construction. The Constraint class implements specific filtering conditions using a ConstraintType enumeration that defines categories like Tag, FileType, and MediaType. Properties are implemented as key-value pairs to store additional attributes that qualify constraints.

The system employs the visitor design pattern through BaseVisitor and its extension SQLBoolExpressionBuilder, which transforms the abstract syntax tree into executable SQL queries. Supporting this structure, FilterState manages query execution parameters such as page size and sorting preferences, while Entry represents the result records returned from queries. The QtDriver class provides a comprehensive interface layer with multiple methods that connect the application to the underlying query system. This class is connected to the querying logic through many other classes not shown in this diagram, which is why it is not connected to the rest of the diagram.



Key changes/classes affected

- models.py
  - Class: Entry
  - For the db in tale entries we added a column to store the aspect-ratio for each image.
  - `aspect_ratio: Mapped[float | None]`
- visitors.py
  - Class: SQLBoolExpressionBuilder
  - Function: `visit_constraint()`
  - Added functionality to filter the images dependent on the aspect ratio inputted by the user.
  - Added following functions
    - `__entry_more_than_aspect_ratio()`
    - `__entry_between_range_aspect_ratio()`
    - `__entry_matches_aspect_ratio()`
    - `__entry_less_aspect_ratio()`
  - Filters are applied to <, >, X-Y and a fixed number.
- enums.py
  - Class: FilterState
  - Added a class method to create a filter state based on a specified aspect ratio. Added `from_aspect_ratio()`.
- ts_qt.py
  - Class: QtDriver
  - Added “aspectratio” to regex and to the list of possible queries
  - Function: `update_completions_list()`
  - Added an autocomplete for “aspect_ratio”: When the user starts to type “a”, “aspect_ratio” pops up as an option.
- ast.py
  - Class: ConstrainType
  - Function: `from_string()`
  - Added constraint for aspect ratio

Architectural overview.

Core architecture

TagStudio is a file organization application designed around tagging with a metadata system to enable easy file categorization and management. The architecture follows a modular approach where there is a clear separation between the back- and frontend.

Key Components

Entry point

The entry point of the application is the tag_studio.py file which initializes the app and processes command line arguments such as –open to open specific libraries and –ui to choose a specific UI.

Backend database

Tagstudio uses SQLite as its database engine storing library data in a .TagStudio/ts_library.sqlite file which is kept in the corresponding library folder.

Core library

TagStudios core library manages file entries, tags and their respective relationships while keeping the original file completely untouched. The Library class acts as the main interface for this as it handles tag creation modification and file system synchronization to automatically update the systems metadata when files are removed, added or modified.

This is all built upon SQLAlchemy ORM with the SQLite database as the structured schema efficiently stores file entries, association etc. ORM vastly simplifies the database interactions by allowing the system to work directly with python objects instead of raw SQL queries which makes maintenance and development easier. Since the metadata is stored in the SQLite database rather than embedding it into the file all of the organizational data is managed independently ensuring non-intrusion and with the benefit of database transactions and backup mechanisms to protect data while preventing corruption or loss.

It also supports library creation, file scanning, tag management, search execution and analytics to top it off all while maintaining high performance even with large libraries.

UI

The main user interface is built using Qt which is managed by the QtDriver class which to no surprise handles all Qt-specific functionality. The benefit of using a driver based system is that it allows for easy integration of future interfaces such as command-line or even web versions without the need to change the core structure.

Tag system architecture

Some of the most important features that TagStudios tagging system offers are:

- Tag objects: each tag has properties such as names, colors, aliases and relationships.
- Parent inheritance: A parent tag allows tags to inherit attributes from parent tags.

- Automatic disambiguation: Duplicate tag names are automatically handled to avoid conflict.

Search architecture

The search system in TagStudio is designed to be flexible and easy to use to enable users to quickly find what they are looking for through the use of search refinement using boolean operators like AND, OR and NOT with added support for grouping and nested conditions. This combined with the tag based search feature allows for full text searching across all names, shorthands and aliases in the current library. In addition to all of this, it also allows users to use special searches like ‘special:untagged’ and our newly added ‘aspect_ratio’ which allows for easy but highly detailed searching.

Design pattern

TagStudio follows a very structured design approach to keep its components modular, maintainable and easy to extend.

The Model-View-Controller, MVC, pattern keeps things clean by clearly separating the different areas of concern. The Model, src.core.library in this case, handles all the data and logic, the View, Qt widgets, handles the UI and the Controller interconnects them ensuring smooth operations between the back- and frontend.

The Factory pattern is used for creating UI components dynamically based on the given command line arguments and for initializing tag objects.

The Observer pattern keeps the UI in sync with the data. Whenever data or tags change everything automatically updates without any need for a manual refresh.

The Command pattern handles user all user actions such as library operations, tagging and searching and through the use of an action history allows the user to undo/redo their actions.

All of these combine to make TagStudio organized, modular, maintainable and flexible.

Overall experience

What are your main take-aways from this project? What did you learn?

We learned to collaborate better with each other, dividing the work, and working on such project as a team. We also learned how open-source projects might work, how the contribution pipeline works, from feature requests, to the integration tools and tests for the feature, and other things involved in the process (reading the documentation to get familiar with the code base, project’s feature roadmap, getting to know the specific tools the project uses, etc).

How did you grow as a team, using the Essence standard to evaluate yourself?

Our team is in the performing state at the moment. Even though we have fallen back last assignment to the earlier stages, now we have figured out efficient ways to communicate and work together, making it possible to achieve this tier. Our work is efficient, we divide it in a way which doesn't interfere with each other, so that each person can do their part efficiently, also minimizing the amount of wasted time.

SEMAT Kernel (point 6 of P+)

The main benefit of the SEMAT is that it can help teams analyze themselves from a more rigorous and objective perspective, ultimately becoming much more efficient in the long run. It facilitates a clear understanding of interdependencies and provides a structured framework for continuous improvement.

For example, the Kernel Alphas describe how the different aspects of working in a software development team are related, which aspects affect each other, and which don't, as well as what each part provides and addresses. They encompass essential elements such as Opportunity, Stakeholders, Requirements, Software System and Work which together form a comprehensive overview of the development process. This can be useful to understand in a more systematic way what the requirements are from the stakeholders, how it shapes the work the team performs, building the system the stakeholders use. Furthermore, it offers a unified vocabulary that enables better communication and clearer alignment between team objectives and stakeholder needs.

Without such an overview of the whole process, much more communication might be needed, and the work a team performs might be much more disorganized and less effective (for example, if the team created its own requirements, they may not be aligned with the ones from the stakeholders, resulting in the work producing a software system which is not very usable to the stakeholders). This lack of coherence can lead to fragmented efforts and misdirected resources and time which can ultimately compromise the quality and usability of the final product. A drawback could be that it often takes a while for a team to learn new ways of working, especially for already formed teams. Adopting the SEMAT framework requires a shift in the way of working and a willingness to accept structured evaluation which may or may not initially slow down progress.

In our case, we're the stakeholders, and as such we dictate our own requirements, and we are held accountable to ourselves. We benefit from doing it right, but being the same means that there's no relationship between the customer and the team working on the product, so there's no other perspective on it other than ours. In addition, since we directly benefit from the resulting product we are incentivized to make it as advanced as possible which may lead to continuous expansion of the project scope ultimately leading to scope creep and overcommitment. Knowing when to quit can become much more difficult due to working in a team where everyone is tunnel visioning on the result.

Even if this may work well in theory, there's often a lot of factors not described by the model, resulting in the team not being able to adapt the ways of working with the kernel effectively. In real-life situations, large parts of the models need to be able to adapt to the needs of the team, stakeholders, and the product itself, where just strictly sticking to the theory might not result in making the process more efficient. Rigid application of the framework without the

necessary customization can hinder flexibility and responsiveness in dynamic and ever changing project environments.

Change takes time and effort, and might not always be worth the investments in it. Moreover, the transition phase may involve a steep learning curve and temporary declines in productivity as the team adjusts to the new methodology. Adapting a model just for the sake of adhering to the principles might also not always be the best alternative, especially if the team is already well functioning with the way of working they have developed. In such cases integrating only the most beneficial aspect of the SEMAT framework may be a more pragmatic approach.

This results in a balance, where one must balance out the possible long-term benefits with the possibility of short-term dysfunctionality of the team, where the work might not be completed with all the resources occupied by the change. Ultimately the decision to adopt or adapt the SEMAT framework should be made after carefully weighing the potential for sustained improvement against the immediate costs of the transition to ensure that the model serves as a guiding tool rather than a rigid mandate of heaven who no one may oppose.