Assignment 4

Group 19

Project

Name: TagStudio

Url: https://github.com/Group-19-DD2480/TagStudio

Project Description:
TagStudio is an early-alpha application built to organize files using a user-defined tag system. The user can create libraries in which to store files, add tags to files and search for relevant tags or information. It works without inherently editing the files, meaning that the tags are not saved using meta-data of the [1]actual files but rather its own database.

Onboarding experience

In assignment 3 we worked with a project called Arviz, unfortunately, the issues for that project were not suitable for assignment 4. The issues were either simple fixes that didn't warrant writing a report, or highly technical issues regarding graphics or statistics that we were not equipped to handle. Instead, we searched for a new project. The onboarding experience with this project has been worse than the previous one. The project has a pretty clear contribution guideline and installation guide, however there were some rather annoying complications with building the project and getting the tests to pass on Windows, even though the project included a build file and documentation. While we did not use it, it's also worth mentioning that the project has an active Discord server that is open to help new users and developers to get everything setup.

Effort spent

1

Member	Ruben	Erik	Tore	Usman	Zarko
Plenary discussions / meetings	1h:Looking at Arviz issues 1h: Looking for other projects 1h: Looking at TagStduio issues 1h: dividing work 30min: scheduling and planning next meeting, dividing work 30min: discussing SEMAT and ESSENCE 1h: sync meeting 1h: retrospective	1h: Looking at Arviz issues 1h: Looking for other projects 1h: Looking at TagStduio issues 1h: Dividing work 30min: scheduling and planning next meeting, dividing work	1h: Looking at Arviz issues 1h: Looking for other projects 1h: Looking at TagStduio issues 1h: Dividing work 30min: scheduling and planning next meeting, dividing work 30min: discussing SEMAT and ESSENCE 1h: sync meeting 1h: retrospective 1h: Rewrite semat	1h: looking at Arviz issues 1h: looking for other projects 1h: looking at TagStduio issues 30min: scheduling and planning next meeting, dividing work 30min: discussing SEMAT and ESSENCE 1h: sync meeting 1h: retrospective	1h: looking at Arviz issues 1h: looking for other projects 1h: looking at TagStduio issues 1h: Dividing work 30min: scheduling and planning next meeting, dividing work 30min: discussing SEMAT and ESSENCE 1h: sync meeting 1h: retrospective
Discussions within parts of the group	30min: discussing how to structure patch				30min: discussing how to structure patch
Reading documentation	30min: reading assignment 30min: reading contribution docs 30min: reading	30min: reading contribution guidelines 30min: reading the assignment 1h: Reading documentatio	30min: Reading contribution guidelines 30min: Reading the assignment	30min: reading contribution guidelines 30min: reading the assignment 30min: Reading	30min: reading contribution guidelines 30min: reading the assignment

	about design patterns 30min: reading qt docs	n 30: Reading (skim reading) code with regards to code comments		documentation 30min: reading qt docs	
Configuration and setup	1h: building on windows 1h: troubleshooting pytest 1h: troubleshooting filepath 1h: building and testing on linux	1h: building on Mac	1h: Try and fail to build on windows 1h: building on linux 30 min: Contact with maintainers of TagStudio	1h: troubleshooting running tests 1h: building on linux	1h: initial building on windows 1h: initial building on linux
Analyzing code/output	30min: reading settings saving and ui code 30min: identifying design patterns 30min: understanding how the new patch affected our code		1h: identifying relevant parts of the code	1h: Analysing the workflow for the settings menu 1h: Analysing the workflow for the preview panel filepath display 1h: Analysing the workflow for the title bar filepath display	1h: identifying the relevant parts for the file attribute assignment 1h: identifying the relevant parts for the main window attributes 1h: looking at the new updates in the main repository
Writing documentation	1h: writing requirements 30min: Summarizing code changes	1h: writing requirements 1h: Writing system architecture 1h: writing system	30 min: working on diagram for system architecture 30 min: Creating	1h: writing tests description	1h: writing requirements 1h: writing about code/class changes

	1h: added section about design patterns	architecture	UML diagram 30 min: Write essence 1h: reading up on SEMAT 30 min: write semat kernel 1h: Writing overall experience		30min: Writing about Essence
Writing code	1h: added file path settings to the settings menu 1h: updating ui after settings change 30min: changed formatting to follow guidelines 30min: trying to resolve merge conflict		1h: resolving merge conflicts due to updated upstream 1h: fix broken code due to updated upstream 1h: resolve merge conflicts due to updated upstream 1h: Prepare repo for PR into upstream	1h: Writing tests for the settings menu 1h: Writing tests for the file path display 1h: Writing tests for title bar update 1h: Troubleshooting problems with opening QApplication using pytest	1h: added file path application to file attributes 1h: added file path application to main window attributes 1h: implement instantaneous feedback
Running code	30min: getting familiar with app functionality 30min: Testing implemented functionality 30min: testing after new patch broke our			1h: Familiarization with the app 1h: Trying to get new tests to work.	1h: user testing and familiarization with the app

	code				
Total	21h	x	20.5h	20h	20h

Overview of issue(s) and work done

Title: [Feature Request]: option to not show full file paths

URL: https://github.com/TagStudioDev/TagStudio/issues/301

URL for PR: https://github.com/TagStudioDev/TagStudio/pull/841

Summary: The problem described in the issue is that the application always reveals full file paths, which can often reveal sensitive information such as the user's username. This is a problem when the user is sharing their screen with others, be it during a Zoom Meeting, presentation at school or work and other similar situations.

Scope: As this is only a visual update, it mostly affects the UI classes which print the library or the current file path. We identified 3 key classes which need to be changed in order for this to work:

1. QtDriver class - prints the library path in the application title, recently opened libraries and a status tip when opening a library
2. FileAttributes class - prints the file path in the file preview panel
3. SettingsPanel class - needs to handle the selection and saving of the user preferences regarding the file name printing
4. SettingItems class - An entry needs to be added for the new settings to be displayed

Requirements

REQ-ID	Title	Description
REQ-001	Toggle for File Display Options	The application should provide a toggle option to either display full file paths or hide them for privacy reasons.

REQ-002	Display Full Path	The application should have a setting for displaying the full file path from the library.
REQ-003	Display Relative Path	The application should have a setting for displaying the relative file path from the library.
REQ-004	Display Only the File Name	The application should have a setting for only displaying the file name.
REQ-005	Update Title Bar	When full file paths are hidden, the application should display an alternative title bar format such as: [library folder]
REQ-006	Update the File Description	When full file paths are hidden, the application should display an alternative file description path format such as: [library folder]/path to selected item/selected item.ext
REQ-007	Remember Selection Choice	The application should remember the user settings when restarting the app

Code changes

- enum.py: SettingItems class: The SettingItems enum provides a centralized and consistent way to reference keys in the settings storage, QSettings. It is used throughout the code to get and set settings. As we're adding a new setting, a new item needs to be added to the enum, SHOW_FILEPATH.
- settings_panel.py: SettingsPanel class: This is a UI element used by the user to change settings. It reads and writes settings to SettingItems interfacing through the driver. A new combobox was added for file path settings.
- file_attributes.py: FileAttributes class: This class takes care of the file description and metadata display. The changes to this class were in the update_stats() function, which previously displayed a full path no matter the user preferences. After the updates, the file path correctly updates to reflect the user preference specified in the settings.

- QtDriver class: This class takes care of the main windows and its menus and status updates. The affected functions were update_recent_lib_menu(), which updates the list of the recently opened libraries, referred to by their paths; init_library(), which opens the library and changes the title of the main window to include the library path; and open_library(), which gave a status update with the path to the library. All of these were previously using the full path to the library and were thus improved by reflecting the user preference specified in the settings.

Patch

Settings Panel:
git diff 4b381e7 297abda
git diff d7bc350 a777b25
File attributes:
git diff 297abda bd73eab
Main Window:
git diff e33ed1a d7bc350

Test Results

Tests written for the new feature successfully pass.

Test	Description	Requirements
test_filepath_setting	This test verifies if the SHOW_FILEPATH setting correctly changes upon applying the File Display Option setting	REQ-001 REQ-002 REQ-003 REQ-004
test_title_update	Verifies that the title bar properly updates depending on the selected File Display Option setting.	REQ-005 REQ-002 REQ-003 REQ-004
test_file_path_display	Verifies if the file description in the preview panel properly updates depending on the selected File Display Option setting	REQ-006 REQ-002 REQ-003 REQ-004

System Architecture

TagStudio is at its core a file/library management application that enables users to organize files and libraries using what is called tag-based system. Tagstudio is created in a way that it does not modify or relocate files but instead maintains a separate metadata database to track tags. The system has a modular, database-driven architecture and can be split into three major identified modules

One of the main identified modules of TagStudio's architecture is the User Interface (UI) which is built using Qt, and it provides a way for users to add, manage and search for tagged files. The UI supports functions such as tag creation, searching by metadata and, with our contribution, customizing how file paths are displayed. It interacts with the backend components to update the tag database, retrieve search results, and provide feedback.

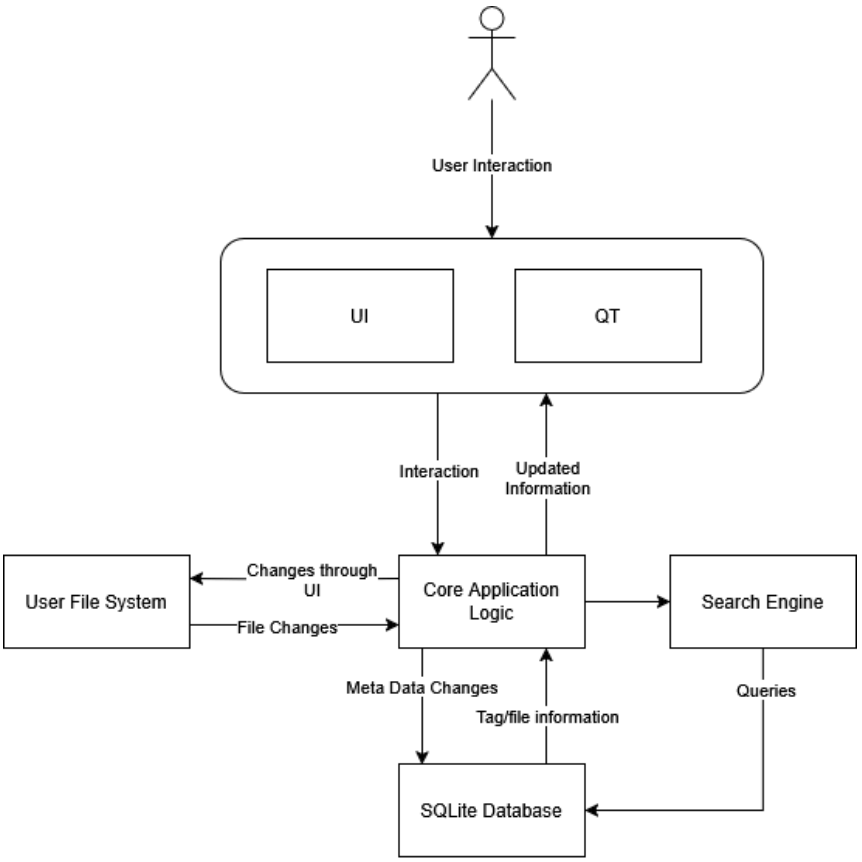
The Core Application Logic serves as the heart of the system which handles user actions and enforcing the rules for tagging, metadata updates, and file tracking. It basically ensures that assigned tags are stored correctly and maintains consistency when files are moved or renamed, and communicates with both the Database Layer and UI to manage file metadata.

TagStudio uses an SQLite database in combination with SQLAlchemy to store the tag-to-file mappings. This Database Layer enables quick searches by indexing tags, file names, and other metadata. It supports structured queries, enabling users to perform advanced searches with for example keyword-based retrieval.

Furthermore there is also a basic Search Engine that enables file searches through in the following formats:

- File entries search based on tags, file path (path:), file types (filetype:), as well as mediatypes (mediatype:) [1]
- Use and combine boolean operators, in specific AND OR NOT, along with parentheses groups, quotation escaping, and underscore substitution to create detailed search queries. [1]
- Use special search variants to find file entries without tags or fields. [1]

The general interaction flow in TagStudio begins when a user for example selects a file and assigns a tag. The UI then forwards this request to the Core Application Logic which updates the Database Layer. At last, the UI updates to reflect the changes.



Design Patterns

Existing System Design Patterns

PyQt: TagStudio uses PyQt for its UI. Qt uses a Model-View-Controller (MVC) pattern, dividing the application into three interconnected components. Model, manages the application's data and logic. View, handles the display and presentation of the data to the user. Controller, interprets user inputs and updates the model and the view accordingly.

UI Elements: A common design pattern for the Qt widgets is the observer pattern. For example the dropdown menus (QComboBox) and the SettingPanel widget. The QComboBox acts as the subject and SettingsPanel listens to changes and reacts accordingly.

QSettings: QSettings is a singleton that sets and gets user settings, keeping the changes between sessions. On linux, the settings are stored in a .ini file in .config.

SettingItems: This uses a constant pattern, a constant enum is created to restrict the possible values for setting names, ensuring they remain within a predefined set. This makes it easier to validate and manage settings.

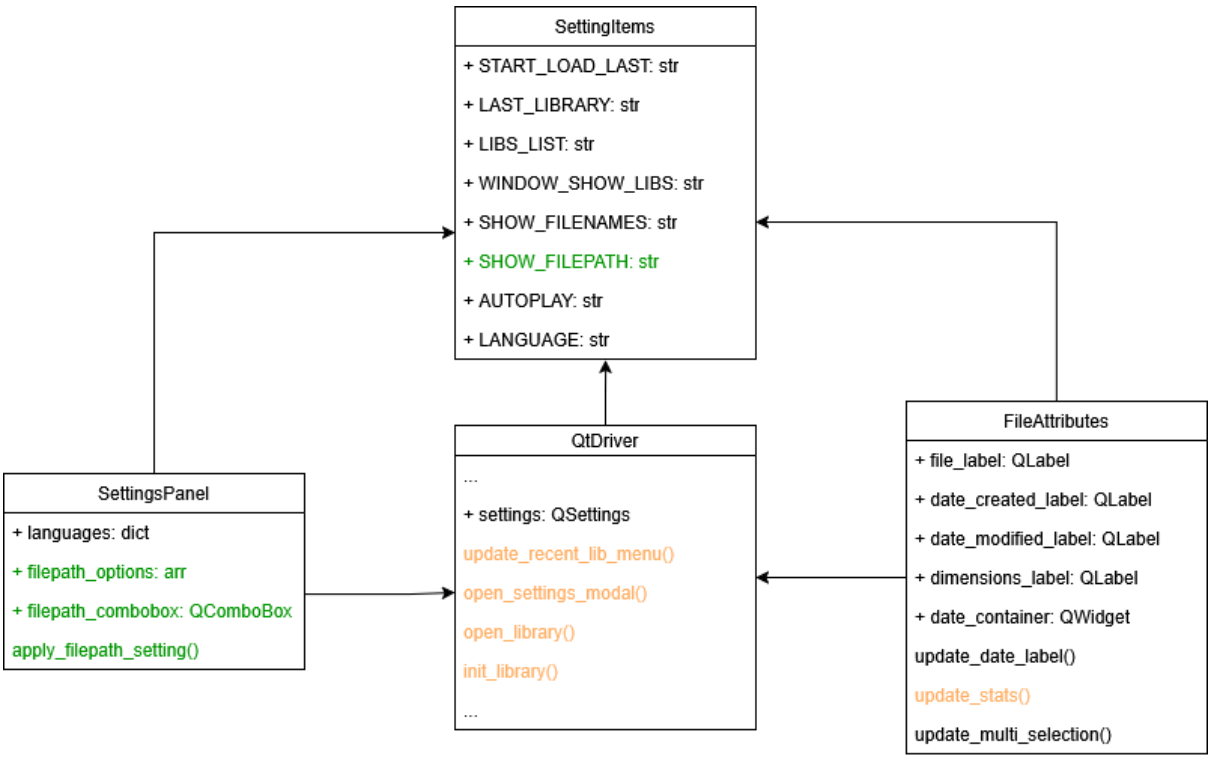
QtDriver: QtDriver uses a factory pattern and is responsible for creating instances of different drivers dynamically. It's used for interfacing with the SQLite database and QSettings.

Database: The database follows several design patterns. The Factory Pattern is used in QtDriver to dynamically create database connections and manage interactions with SQLite. The Proxy Pattern is present in the database access layer, where objects like Tag acts as an intermediary between the UI and direct database queries. The command Pattern is applied when executing SQL statements like INSERT, SELECT and DELETE, encapsulating operations as discrete commands. Lastly, the Iterator Pattern is used when fetching query results with functions like fetchall().

Newly implemented design patterns

Our implementation of displaying the path follows a **Strategy Design Pattern**. This design pattern aims to reduce the size of the codebase, and thus reduce the possibility for errors, by giving functions multimodal functionality based on the selected "Strategy". In our implementation, the strategy is the file path display option, which is first set by the user in the settings and then later followed by the various UI elements that display the filename. The UI thus decides at runtime which path mode is to be used and dynamically changes the path display, without having to create separate functions for each of the modes.

UML Class Diagram



Key changes/classes affected

There are no new classes being added. The updated classes are SettingsPanel, SettingsItems, QtDriver and FileAttributes. In the two settings classes, new functionality is being added as the program did not support filename preferences before. QtDriver and FileAttributes classes do not get any new functionality but their functionality is updated to follow the new options. More in-depth description of the changes is in the Code Changes section.

Overall experience

In large, the most difficult part of this assignment was to find a good issue to work with. Many issues, especially ones in the previous project we worked with, were either incredibly technical bugs or very large feature requests. A lot of time was thus spent trying to find an appropriate project which had a good balance of smaller feature requests that could be implemented in an appropriate time frame. After finding a good project, it was smooth sailing however. The feature we implemented turned out to be a very good fit for the assignment in both complexity level and scope. The project also seems

very friendly and open to beginners which made the contributing aspect not as daunting as we felt we had a bit more freedom when it came to details. Figuring out the architecture of the system, where to make changes etc was a bit tricky since there was no documentation for the backend, but the system was small enough that we quite quickly managed to find it by searching around. One problem we ran into while working on the new feature was that upstream moved incredibly quickly. Overnight we would see 6-8 new commit that often affected the same parts as our own feature. This led to a lot of merge conflicts that were not always the easiest to resolve since we also had to update our own code to be inline with the new standard.

Essence

At the moment, the team would argue that we are in the 'Performing' stage. We work together as a cohesive unit, with defined goals and a high degree of trust. We fulfill all of the requirements for the 'Performing' stage without fulfilling any of the requirements of the next stage, 'Adjourned'.

- ☑ ~~The team consistently meets its commitments.~~
- ☑ ~~The team continuously adapts to the changing context.~~
- ☑ ~~The team identifies and addresses problems without outside help~~
- ☑ ~~Effective progress is being achieved with minimal avoidable backtracking and reworking.~~
- ☑ ~~Wasted work and the potential for wasted work are continuously identified and eliminated.~~

The first requirement is backed up by our previous assignments which were approved by the course authorities and by the internal goals and deadlines set forth by the group.
The second requirement is also backed up by our previous assignments.
The third requirement is backed up by our Drive directory and Discord server, where one can see that no outside help was relied upon.
The fourth and fifth requirements are backed up by our meetings, where we split up the work in an effective way, which minimizes the risk of wasted work.

SEMAT Kernel

Stakeholder

- In the case of assignment 1-3, no customer exists since we produce the code for ourselves
- The examiner, while considered a stakeholder, does not have a direct gain from our success (as in financial gain etc), but still have to work to support us
    - Feedback occurs in a more binary manner, by passing or not passing us
- For assignment 4, the maintainers/users of the FOSS software should also be included.

Opportunity

- Opportunity exists not because of the market or financial opportunity, but because all of the team members take the course which creates the reason for the work to be carried out.
- Solution is created for an artificial problem.
- Means risk and ROI are not of concern
- For assignment 4, the opportunity arises to change an existing system through the opportunity provided by the course.

Software System

- The platform (python) was chosen mainly because it's what the members were comfortable with, rather than to fulfill requirements
- Due to the tight deadlines, the creation of the software often had to be quicker than what would be expected in a real world scenario, more faults were accepted as ok and less thought was put into long term maintainability rather than to get the thing through the door.

Requirements

- Long-term maintainability was not relevant for assignments 1-3 while it would be a massive deal in a real world environment.
- Requirements are more related to completing a task than creating something useable (again different for assignment 4)
- Requirements do not necessarily come from the stakeholders since stakeholders do not have demands

Work

- A lot of steps could be skipped as requirements, constraints, funding, time commitment etc were not of concern.
- Work culture and structure is different than in a real world context simply due to the freedoms given by the team being students
    - Work when it fits rather than 9-5
    - Communication occurring in a more relaxed manner

Team

- Team was fixed rather than being flexible and changing when bringing people on/people leaving.
- Team worked together during a much shorter period of time than a real team would. The team spent less time in a highly performing state due to a lack of time to get used to working together.

Reference List

[1] TagStudio, "Tagging & Metadata Fields," Accessed: Mar. 1, 2025. [Online]. Available: https://docs.tagstud.io/#tagging-metadata-fields