Deggendorf Institute of Technology
Faculty of Computer Sciences
Studiengang Master in Applied Computer Sciences

Automatic source code analysis of projects using the Gitlab API

Masterarbeit zur Erlangung des akademischen Grades:
Master of Science (M.Sc.)
an der Technischen Hochschule Deggendorf

Vorgelegt von:
Sandesh Gharge
Matrikelnummer: 00821875

Prüfungsleitung:
Prof. Dr. Udo Garmann

Am: 08. November 2023

TECHNISCHE HOCHSCHULE DEGGENDORF

Erklärung

Name des Studierenden: Sandesh Gharge

Name des Betreuenden: Prof. Dr. Udo Garmann

Thema der Abschlussarbeit:

Automatic source code analysis of projects using the Gitlab API .....

.....

.....

.....

1. Ich erkläre hiermit, dass ich die Abschlussarbeit gemäß § 35 Abs. 7 RaPO (Rahmenprüfungsordnung für die Fachhochschulen in Bayern, BayRS 2210-4-1-4-1-WFK) selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

   Deggendorf, ...
   Datum Unterschrift des Studierenden

2. Ich bin damit einverstanden, dass die von mir angefertigte Abschlussarbeit über die Bibliothek der Hochschule einer breiteren Öffentlichkeit zugänglich gemacht wird:
   - ◯ Nein
   - ◯ Ja, nach Abschluss des Prüfungsverfahrens
   - ◯ Ja, nach Ablauf einer Sperrfrist von …Jahren.

   Deggendorf, ...
   Datum Unterschrift des Studierenden

Bei Einverständnis des Verfassenden vom Betreuenden auszufüllen:

Eine Aufnahme eines Exemplars der Abschlussarbeit in den Bestand der Bibliothek und die Ausleihe des Exemplars wird:

- ◯ Befürwortet
- ◯ Nicht befürwortet

Deggendorf, ...
Datum Unterschrift des Betreuenden

Abstract

This thesis focuses on the automation of the tasks of evaluating a wide range of projects by professors and developers. The evaluation process is a laborious task of rigorously analysing several projects for the extraction of project summaries and code reviews. Details extracted are further used to grade the project submitted by a student. In today's world, accomplishing all these manual tasks can be replaced by the usage of predefined libraries for the automation of such processes. To find this solution, I commit myself to developing a software tool designed to expedite and optimize the project evaluation process and student grading under the guidance of Professor Udo Garmann.

The first step in this attempt is to gain an extensive understanding mode of submissions and how mentors and professors handle a large number of submissions every year in all Universities across the globe. Later we will discuss the challenges that professors frequently face during project evaluation. The knowledge gained will serve as the foundation for developing an efficient and effective tool design. The findings of this research will be essential in identifying the most suitable type of software to serve the purpose.

We will proceed to determine the best platform for software development that is loaded with rich libraries and compatible with multiple platforms. This compatibility will help the developer maintain a single code base and easy maintenance of the upgrades. The developed software will be rigorously tested to create a robust and dependable tool that can help the project evaluation and significantly save the professor's or developer's time for analysing the project details for further respective purposes.

Contents

1 Introduction

In today's rapidly evolving IT landscape, the pace of software and technological advancements is nothing short of astonishing. Professionals across the industry are investing tireless efforts to keep themselves updated and strive to be at the forefront of innovation. Even students are wholeheartedly dedicating themselves to attaining the knowledge and skills necessary to meet the industry's demands. In this dynamic environment, the role of professors and mentors is paramount in guiding and educating students to navigate the intricate realm of information technology.

Within the academic sphere, universities face the formidable task of managing a burgeoning number of students each year. This management encompasses the meticulous review of assignments, assessments, and, notably, final projects. These projects, a common platform for showcasing students' capabilities, are typically maintained and tracked using Version Control Software Tools such as GIT, CVS, SVN, and others. Among these tools, GIT stands out as one of the most popular choices for version control due to its powerful features and widespread adoption.

One of the significant challenges faced by academic managers and professors is effectively maintaining and monitoring the GIT dashboard, which provides a comprehensive view of the projects they are involved in. For professors, this list of projects is extensive and ever-expanding, evolving with each passing semester. Here we put forward the first research question as *What are the key challenges faced by professors and academic managers when handling a significant number of student projects, and how can technology mitigate these challenges effectively?*

This thesis aims to address these challenging issues by presenting a strong, digital solution. To cope manually with increasing number of students and an expanding collection of project work in any academic institution across the globe effective management tool becomes necessity. With the availability of development platforms and libraries, it is possible to mitigate such manual processes and provide a solution to improve the academic experience. Under the guidance of Prof. Udo Garmann, we dedicate our efforts to coming up with a tool that can not only help the professors or evaluators complete their tasks at a better pace but also improve the academic experience with an automated approach. It will be followed by discussing another research question *What is Static Code Analysis? Can Static code analysis help professor while evaluation of the projects?*

Static Code Analysis tools efficiently examine the code of a program without executing it. It is an extremely helpful tool to identify the violation of coding standards, its potential issues and vulnerabilities. This analysis proves helpful in code maintenance, program quality checks and enhanced security. As a part of the research question, we look forward to utilising the results examined by such tools for the evaluation of the academic projects. The evaluator will have ample details for grading such projects without even executing them.

This software is more than simply a digital tool which represents an approach that prioritizes learning, creativity, and student potential in the educational process. It will relieve instructors of administrative complications, allowing them to focus their efforts on the educational enhancement of the students. As a consequence, students will be able to develop, explore, and create under the supervision of mentors who have the time and resources to give a truly rewarding educational experience.

2 State of Art

The number of universities in the world is excessive, and each semester, a large number of students dutifully submit their projects for evaluation. This process involves meticulous maintenance, evaluation, and subsequent grading by professors, and it forms a fundamental part of the academic experience.

Managing and evaluating such projects is a difficult and diverse process. It includes components such as tracking project progress, delivering timely feedback, and keeping records.

There are several challenges in this approach. Because of the overwhelming amount of projects, a strong system that can handle the flood of assignments is required. Given the workload, providing timely feedback, which is critical for student improvement, can be difficult. Furthermore, keeping project data secure is essential.

As technology and IT processes evolve, there is a growing demand for simplified solutions that may relieve professor's workloads and improve the entire academic experience. In this setting, the development of software tools to aid in project management and assessment becomes essential. Such technologies can help academics traverse the difficulties of project evaluation more efficiently and effectively, benefiting both professors and students.

In the following sections, we'll go deeper into the complexities of these issues, looking at possible solutions and the implications for the academic community. We'll talk about the importance of version control systems like GIT in this context, as well as how technology may be used to bridge the gap between academic needs.

2.1 Mode of Submissions

In reaction to the ever-changing digital world, the submission of assignments, projects, and coursework has experienced major modification in today's academic the situation. Traditional means of submission, such as hard copy delivery in person, have given way to more convenient and effective digital ways. In the past, students would personally show their work or submit printed copies for review. However, due to the availability of digital alternatives, such in-person submission has becoming more rare.

Email submission was one of the first digital submission techniques, and it was mostly utilized for document-based assignments. When it came to contributing code or projects with a large number of files, however, email proved challenging. Compressed files were frequently used as an improvised method for sending larger and more complicated projects.

To ease assignment and coursework submissions in today's academic scene, colleges have implemented online platforms such as Learning Management Systems (LMS) such as Blackboard, Canvas, Moodle, and institution-specific platforms. Students may easily upload files, papers, or projects to specific course sites inside these LMSs. Professors and teaching assis-

tants may then access, evaluate, and grade these contributions online, making the assessment process more efficient.

Cloud storage solutions like as Google Cloud, OneDrive, and Dropbox have developed as popular submission mechanisms in the lack of specific LMS sites. Students may use these platforms to securely store and exchange project files, promoting accessibility and cooperation.

Version control systems, such as GIT, have become essential for coding projects. GIT repositories are developed to hold project-related work, allowing instructors to swiftly analyze, evaluate, and assess submitted code and project components. The variety of different submission techniques underscores the dynamic nature of academic evaluation, underlining the necessity for creative ways to successfully manage and analyze unique student work. In the parts that follow, we will look deeper into the issues and solutions associated with academic project management, with a special emphasis on GIT as a critical tool for dealing with coding-intensive projects.

2.2 Challenges faced by Professors

It is critical to obtain a complete grasp of the difficulties that affect this process before designing and developing software solutions to meet the obstacles experienced by professors in handling a high number of student projects. This understanding serves as the foundation for developing optimum features and successful solutions.

2.2.1 Maintaining large number of projects

When submitting work for assessment or review, students frequently include professors as project participants on systems such as GitLab. When academics connect into their Git accounts, they are greeted with a large number of projects. This catalog includes not only projects from the current semester, but also projects from prior semesters and any personal initiatives undertaken by the lecturers. Navigating this ever-expanding list to find a specific project is difficult, making project management using GitLab a time-consuming task.

2.2.2 Manual Evaluation of projects

One of the most important tasks for instructors at the end of each academic semester is to evaluate student-submitted projects. All the instructors must conduct a thorough analysis of all project features. This comprises manual code error-checking, comment-quality evaluation, file enumeration, and consideration of other critical criteria. While human execution of these activities is traditional, it is time consuming if compared to newer methodologies for the incorporation of prepackaged libraries, allowing for a more efficient and automated review process and thereby expediting a piece of the assessment procedure.

2.2.3 De-Registering from projects

Accessing the settings of individual projects and de-registering as a students from them is one proposed solution to addressing the difficulty of managing a large number of projects.

This step essentially removes the project from the professors' dashboard, resulting in a more concentrated presentation of relevant initiatives. However, given the large volume of projects, manual implementation of this solution might be time-consuming and labor-intensive.

These highlighted challenges serve as the foundation for our study, which is devoted to the creation of a software solution intended not only at reducing these difficulties but also at improving the whole project management and assessment process for academics in the academic setting. In the following parts, we will look into the platform selection for software development as well as the strategies used to overcome these difficulties.

2.3 Selection of Platform for development

2.3.1 Type of Application

As a solution to all the problem stated, we have 2 major choice to design an application. Selection of type of application will depend upon the requirements of the professors and their feasibility. Before we start the implementation, we will first discuss the major type of software and their features, followed by the requirements that can help professor perform their task swiftly. This background will help us in the selection of the type of application. Lets go into details about the major type of application we see in today's world. It can be -

Web Application

Web application is he application hosted on the internet and accessible from anywhere around the globe. It follows a client-server architecture. The client (the user's web browser) sends requests to a remote server, which processes those requests and sends back the necessary data or content. This separation of concerns makes it easier to manage and update the application. When it comes to implementation of our problem using web application there are few things to look for before we select the platform.

Advantages -

- We have extensive amount of platforms and frameworks to design a Web Application.
- Once we design and develop the application, the approach will be to access this application via Internet and possibly connect internally to GIT using APIs.
- Evaluation can be stored in the structured or unstructured database and can be retrieved on demand.
- Once hosted, the application can be accessible from any device using browsers.

Disadvantages -

- At least 3 layer architecture is needed for a Web Application to run effectively
- Web Application needs generally needs an internet connection depending upon the complexity of the application

- One must have good knowledge in the field of front end, back end as well as database tools

Desktop Application

Desktop application brings another set of flexibility when compared to Web Application. It is a computer program that runs on user's personal computer. Unlike web applications that run in web browsers, desktop applications are designed to run directly on the user's operating system. Lets discuss few points about desktop applications -

Advantages -

- Similar to Web Application there are many platforms in popular languages for developing a Desktop Software.
- Once installed into a system, it can be used without internet connection.
- Though desktop application can work offline, internet connection can be used to connect to GitLab repository using APIs if needed.
- Generally, one programming language or platform is enough for implementing proposed design.

Disadvantages -

- Being a desktop software it can be accessed only from system where they are installed.
- Development may vary according to the operating system leading different versions of code for each type of operating system.

2.4 Requirements

After having a productive meetings with professor Udo Garmann, we were able to list down all the requirements. This list will lead us to the further selection of platform and language that we will be using for development of the application.

1. Get details from readme.md file e.g. Name, path of readme file, other links such as link to project repository in GIT.
2. In case of presence git link in the readme file, clone the project for analysis.
3. Option to analysis the project downloaded in the compressed form (.zip).
4. Group projects according to Course Name
5. Analysis of the project should contain few specific details
   a) Total number of lines
   b) Total number of Comments

c) Total number of files
d) Total number of Source files
e) Details of different types of files
f) Analysis of Source Code files

6. Window to grade the project
7. De-registration of the project
8. Display average grades post grading of the project with different parameters
9. Preferring tool which can work offline
10. Post evaluation, save the data into .xlsx or .csv file

2.5 Proposed Solution

Since we are ready with the requirements to design the tool, in this section we will talk about the selection of software type and platform of implementation.

2.5.1 Type of Software

Referring to previous section of requirements, point number 9 states preference to an application which can run offline. Out of the two choices that we have i.e. Web Application and Desktop Application, the obvious selection would be desktop application. This will help professors to evaluate the projects offline and also in absence of internet connection. Evaluated data is demanded to store in form of files as discussed in the point number 10 of previous section.

2.5.2 Platform of Development

When it comes to designing and development of software, numerous platforms are available for desktop application development. Selection of platform for desktop application development is also challenging task due to availability of many number of frameworks that are well structured, compatible with all operating systems and can provide ample amount of features and libraries to implement proposed design effectively. Out of all the frameworks, selection of PyQt6 Framework for software development was the favourable choice. PyQt6 is a python based framework, hence, the developed software will be compatible with all the major operating system. We will talk about the further advantages in the Background section which will give clarity to the selection.

2.6 Can automatic Static Code Analysis contribute in evaluation ?

All the points in requirements section can prove useful for evaluation of any project or submission for professor. But this data can be insufficient for evaluation, as the code written by

the students will vary. Along with other factors it becomes important to understand if the code written by student is efficient, has followed proper coding standards, if it is error free or if the application is secure enough. These are few points which can prove deciding factor if it comes to check the quality of the software and process to detect such common quality checks termed as Static Code Analysis.

2.6.1 What is Static Code Analysis?

Static code analysis is analysing the code without execution of the actual application program. Static code analysis gained its significance in the 1970s where formal review and inspections were recognized as important to productivity and product quality, and thus were adopted by development projects [1]. This analysis at early stages of software development proved useful to produce more reliable and efficient programs.

Analysing static code in initial days was manual which is time consuming. Effective analysis needs all the information regarding type of errors an auditor needs to find before rigorously examining the code [1]. Apart from auditors, manual review can also be performed by the developer which is termed as self-review. The peer review is when the programmer presents his code to a colleague for review. But there are static analysis tools which compare favourably to manual reviews because they are faster leading to frequent evaluation of programs anytime. These tools look out for fixed set of problems and cannot solve all the security problems or cannot have same level of expertise as a human auditor has. Static code analysis works differently for types of programming language, hence with respect to static code analysis the programming language can be separated into two types of programming languages namely General-Purpose Programming Language (GSL) and Domain Specific Programming Language (DSL) [2].

A GPL is a computer language that is broadly applicable across application domains and lacks specialized features for a particular domain. Static analysis tools and quality inspection tools, in general, are prepared to analyse the source code of GPLs. Contrary, a DSL is specialized in a particular application domain. The line is not always sharp, as a language may have specialized features for a particular domain but be applicable more broadly, or conversely, may be capable of broad application but in practice is used primarily for a specific domain [2]. Simple example of languages that can distinguish between GPL and DSL are Python and SQL. Python is a language which is used across various application domains whereas, SQL strictly works specifically with DBMS. In this thesis the scope of static code analysis is confined to GPL as it will cover all the programming languages which are necessary for development of any application which will be majority of the submissions by student for evaluation.

2.6.2 Use Static Code Analysis for Evaluation

As we learnt, static code analysis tools can statically analyse the code to find bugs, security vulnerabilities, security spots, duplication and code smell. Hence we can say this analysis will give a brief information on the quality of the code that has been analysed and how it can help a developer to refactor the code to eradicate all the bugs and vulnerabilities. This thesis aims at using the results produced after analysis for evaluation of the application code.

Professor or evaluator can find these information useful to judge the code considering specific parameters. Use of automatic code analysis can save remarkable amount of time for evaluation when compared to manual analysis. The tool that we will be using to analyse the code is Lint. More information about the tool is explained in detail as we proceed to the next chapter.

3 Tools used for development

To understand entire project, one must know all the tools used during the implementation. This project revolve around one language i.e. Python. This section mainly describes about the python libraries used for software development.

3.1 Python

Python is a popular language in almost every field of IT world, whether it be Web Development, Data Analysis, Artificial Intelligence or Scientific Computing and many more. As a historic background, it was introduced by Guido van Rossum (BDFL) in late 1980s. Since then, we can find Python as widely-used high-level programming language for its flexibility, simplicity, and readability.

Python have provided immense libraries for developers to write concise as well as efficient code which leads to rapid development and easy maintenance of any application. Ability to write program in any programming paradigms i.e. procedural, object-oriented or functional programming enables developers to loop in Python in any project requirements. Out of all the libraries, few of them are used in this Software Development project namely pathlib and Pandas which are further discussed in detail.

Furthermore, Python allows applications to run seamlessly on multiple operating systems, including Windows, macOS, Linux, and more. Its cross-platform compatibility and open-source nature encourages a vibrant community of developers to contribute to the growth by continually expanding its capabilities and versatility. All these reasons make Python a preferable choice of programming language. We will now go through libraries that were used in the project in the next sub sections.

3.1.1 pathlib

The software has a plethora of capabilities designed to provide a detailed analysis of various project kinds. This examination includes a thorough examination of file numbers, file kinds, and the inclusion of comments inside. The Python package 'pathlib,' which addresses the complexities of file path and file system operations, is useful.

'pathlib' efficiently alleviates the difficulties associated with managing file paths by simplifying basic operations such as path amalgamation, resolution, directory traversal, property validation, and file interaction. Its use results in code that is not only beautiful but also free of the frequent problems that result from traditional string-based path manipulation.

In conclusion, 'pathlib' proves to be a helpful asset, particularly for those involved in file-related tasks within Python applications. It implements a modern and streamlined technique

that improves the effective administration of both paths and files within the application framework.

3.1.2 Pandas

Pandas is Data Analysis library useful for professionals working in the field of data analysis, machine learning, finance, research, and more. The name Pandas is **derived from the word 'Panel Data' i.e. multidimensional structure data sets. The main purpose of using this library is to display information of projects and details related to it. This data structure can be used to save the details in .csv or .xlsx format as per the preference of the user using predefined functions.

3.1.3 pygount

While analysing software or web Application projects our few of the task would be to extract information such as the type of files, number of lines of code written, number of comments added, show total number of files and many more. Python has a predefined library pygount which has few predefined functionality that will help us get information regarding such projects.

pygount supports multiple programming languages and can swiftly compute even large projects. It can be used in command line as well as inside the code. The results obtained using this library can be fetched in different formats such as JSON, xml or even in plain text. pygount can provide a detailed breakdown of line counts, including the total number of lines, blank lines, comment lines, and code lines for each specified language. Also being an open-source project, it is has benefits of updates and improvements with the time.

Usage of pygount has significantly reduced the time of development and helped us to write a concise and efficient code.

3.1.4 pyinstaller

pyinstaller is widely used essential utility in Python ecosystem converts a Python application into executable files for multiple platforms like Windows, Linx, MacOS. It analyses the python code and automatically identifies the necessary dependencies including third-party libraries and bundles them into the executable. All the process is performed on single base code hence, developer do not have to work on re-writing the exiting compatible to multiple platforms. More over pyinstaller is a open source and freely available with a vibrant community to find solutions and support for common issues. Overall it significantly reduces the efforts for final deployment of an python application.

3.2 Lint

Lint term used in computer science for a tool which analyses the static code to detect programming errors, bugs, coding standards violations and for many other purpose. 'linter' is the term used for program which performs the function. Linter identifies potential issues in the code without manual inspection which could help professor to get an idea about the quality of the

code in the submission. Linter can also detect compilation errors which can related to syntax, undefined variables or type mismatch. It can analyse wide range of programming languages including Python, C/C++, Java, JavaScript and more. This thesis will work with linter used to analyse Python and JavaScript code using libraries elaborated as below.

3.2.1 pyjslint

pyjslint is a Python package that can be used to run JSLint directly from a Python environment, allowing analysis of JavaScript code without leaving the Python ecosystem. Indirectly it provides a quality check for JavaScript code into Python-based workflow or projects. This library proves really useful for analysing the projects based on JavaScript frameworks. This is initial phase of the software and later more Lint packages can be incorporated for analysing projects based number of platforms.

3.2.2 pylint

pylint is a Python package that can be used to check for code analysis of python files. This library performs static code analysis without executing it and catches errors without even executing the files. Here this analysed data will be used by professors for grading purpose of the submissions. Being open source and freely available, pylint is actively maintained and updated along with python updates and coding practices.

3.3 PyQt6

Developing a desktop software has a major challenge of writing code for three different major platforms i.e. Windows, Linux and macOS. PyQt6 is a powerful and widely used Python framework which not creates desktop application with graphical user interfaces (GUI) but also allows to design cross-platform applications that run seamlessly on major operating systems as listed above.

PyQt6 offers extensive collection of UI components and widgets that helps is building versatile and interactive GUI. It is powered with mechanism of Signals and Slots which enables easy communication between components and modules of the software. It also has a platform that can be used to design GUI using tool named QT Designer using simple drag-and-drop interface and export them to Python code for adding back-end functionality.

We will talk more about the design and implementation of the PyQt6 in detail as move ahead. Classes defined in PyQT6 will be introduced as required in the later sections. This is way it will be easy to understand and navigate,rather than jumping to and fro from different sections to this section.

3.4 Visual Code Studio

Selection of IDE will not affect the quality of software but will provide ease of development for the developer. IDE which has enough features which consumes less system resources would be an ideal choice. Visual Code Studio not only has rich language support but provides consistent

development experience across different Operating System i.e. Windows, macOS, Linux. Its vast market place of extensions covering wide range of areas such as language support, debugging, source control, and more. VS Code offers a Live Share extension, enabling real-time collaboration with others. It allows multiple developers to work on the same codebase simultaneously, making pair programming and debugging more effective. Getting hands-on with such IDE can be an advantage for the developer as this is popular tool for Web development, Mobile app development, Back-end development, Data Science and AI.

3.5 GIT

Developing any kind of software or application needs a version control system for maintaining different versions of code, allowing multiple developers to work together as a team. GIT was the preferred choice as it is one of the most popular open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

3.6 Latex

Following the successful completion of software development, the documentation for this thesis was meticulously prepared using the LaTeX typesetting system. LaTeX is a widely recognized and extensively used tool, mostly within academic, research, and publishing domains. It is know for its exceptional support for mathematical and scientific content, as well as its proficiency in generating aesthetically pleasing documents, which also includes academic papers, theses, books, articles, and presentations. Notably, LaTeX is an open-source and freely available technology that facilitates cross-platform compatibility and smooth editing using any text editor. This thesis was written using an online LaTeX compiler, which can be found at https://sharelatex.th-deg.de. This platform considerably took care of document design by including features such as page breaks, line numbering, and indexing, significantly easing the documentation process.

3.7 Agile Methodology

Agile methodology is a popular framework used globally for any software development cycle. It ensures regular development, user feedback, track the progress and make necessary manipulation in the targeted goals in case of product back logs. Daily short meets and division of bigger task into sprints that proves helpful in tracking the progress on frequent basis gives a good idea for customer about the progress. Along side all the features, there are also privileges for product owner to alter the requirement as per dynamic needs are also considered during the development phase. It surely work with the team but can also be used in case of individual developer. Agile methodology is used in this thesis not only for software development but also for entire process followed during research followed development and testing.

There are few popular Agile Frameworks that are preferred for any software development e.g., Scrum, Kanban, Extreme Programming (XP). Scrum framework is structured framework of

all with defined roles e.g. Scrum Master, Product Owner and Development team, defined ceremonies like Sprint Planning, Daily Stand-up, Sprint Review, Sprint Retrospective, and artifacts namely Product Backlog, Sprint Backlog, Increment. Whereas when it comes to Kanban, its focus is narrowed down to continuous flow without any defined roles and ceremonies. A Kanban board is used for continuous flow where the task defined are in 'To-Do' column, later moved to 'Done' once completed. Extreme Programming is preferred framework for delivering a product developed with excellent coding quality, best engineering practices and strong customer collaboration. Hence, depending upon the flow one has to decide the choice of framework suitable for the thesis.

4 UI Implementation

The desktop application that we are about to design is divided into 2 windows. Main Window and Analysis Window. Prof. Udo Garmann have already worked on developing the main window. To carry forward, we will talk about how analysis window will help professor evaluating the projects submitted by a student.

4.1 Proposed Design

There are two windows which will work together to complete the software application i.e. Main Window and Analyser Window. We will discuss UI for each each window separately in detail.

4.1.1 Main Window

The image displays the main window of the software. It is divided into different section and with each section have its own significance. We will talk about each section in detail.

Figure 4.1: Dashboard of the software

Menu Bar

Menu is most common feature of any desktop application. There is only option i.e. 'File' which displays 3 options on click. Upcoming versions will have more options with updated features.

1. Load iLearn Zip - This option can be used to load the project stored in the zip format
2. Save Overview Report - Report can be saved either in .xlsx or .csv format using this option
3. Close - On click of Close button, application closes after warning message is displayed on the screen

All the options are loaded with shortcuts as shown on the right side of the button.

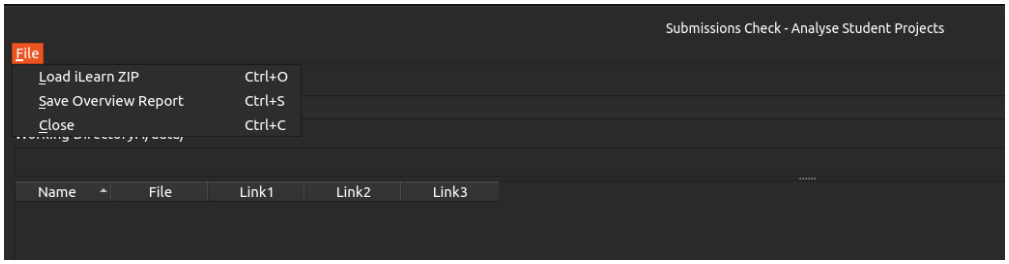

Figure 4.2: Menu Bar

Course Name

This section is a combination of static label along with an edit text field on the next line. This edit field can be used to group projects of the same batch of students. e.g. mit-ws-22-23 stands for the course name Modern Internet Technology (mit), Winter Semester (ws), followed by year of semester i.e. 22-23.

This field is always editable and can be customised when needed. A folder is created inside the output directory with the same name as the value present in the text field and store all the projects.

Figure 4.3: Course Name

Access Token

Access token is necessary only in case the user wants to de-register from the project on Gitlab. Access token acts as a Authenticator for the user to take various actions on git repositories using Gitlab APIs. Each account will have unique access token that has to be generated with defined access for manipulation.

This field is always editable and the text entered is not visible to keep the access token secure, similar to a password field.

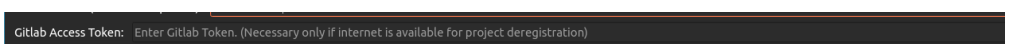

Figure 4.4: Enter Access Token

Working Directory

This field is a non-editable label, that gives the information about the current path where all the projects are present.

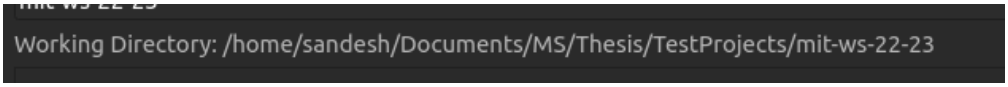

Figure 4.5: Working Directory

Project Details Table

Second last section of the dashboard displays the data of project in the form of student's name, overall grade, path of ReadMe file and the links present in the Readme file. There are multiple links displayed for a given table. In case one of the link is a link to git repository, it is easy to clone the project just on clicking the link cell in the table.

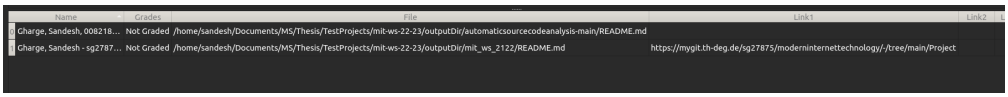

Figure 4.6: Project Details Table

Display Panel

After the project is loaded and if we click on the path of the Readme.md file, the display panel display all the content of the ReadMe file.

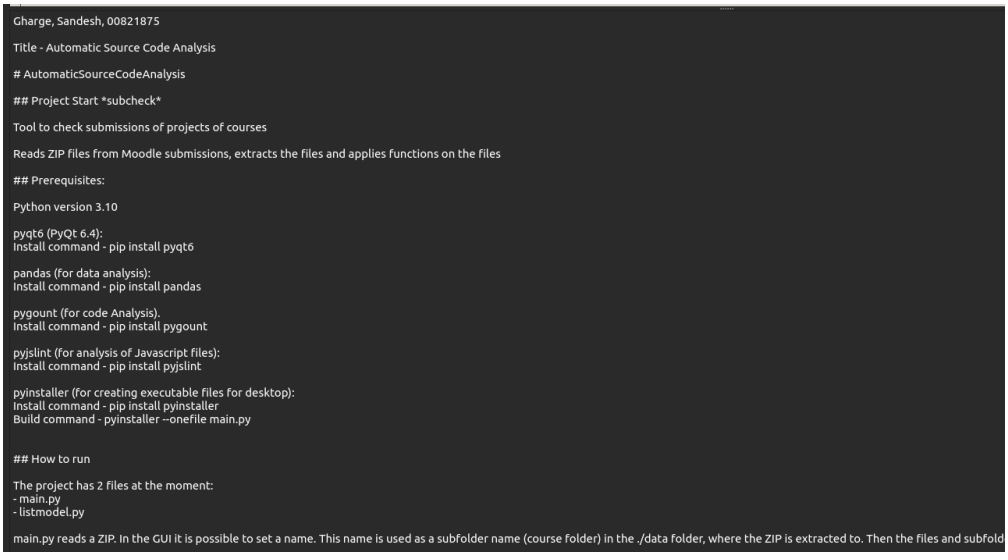

Figure 4.7: Display Panel

4.1.2 Analysis Window

After we upload the project in the main window, Analysis Window is designed to analyse, grade and de-register from the project. Analysis window has the name of the student at the top referring to the student to whom the project belongs to. Each task is divided into different tabs as shown below -

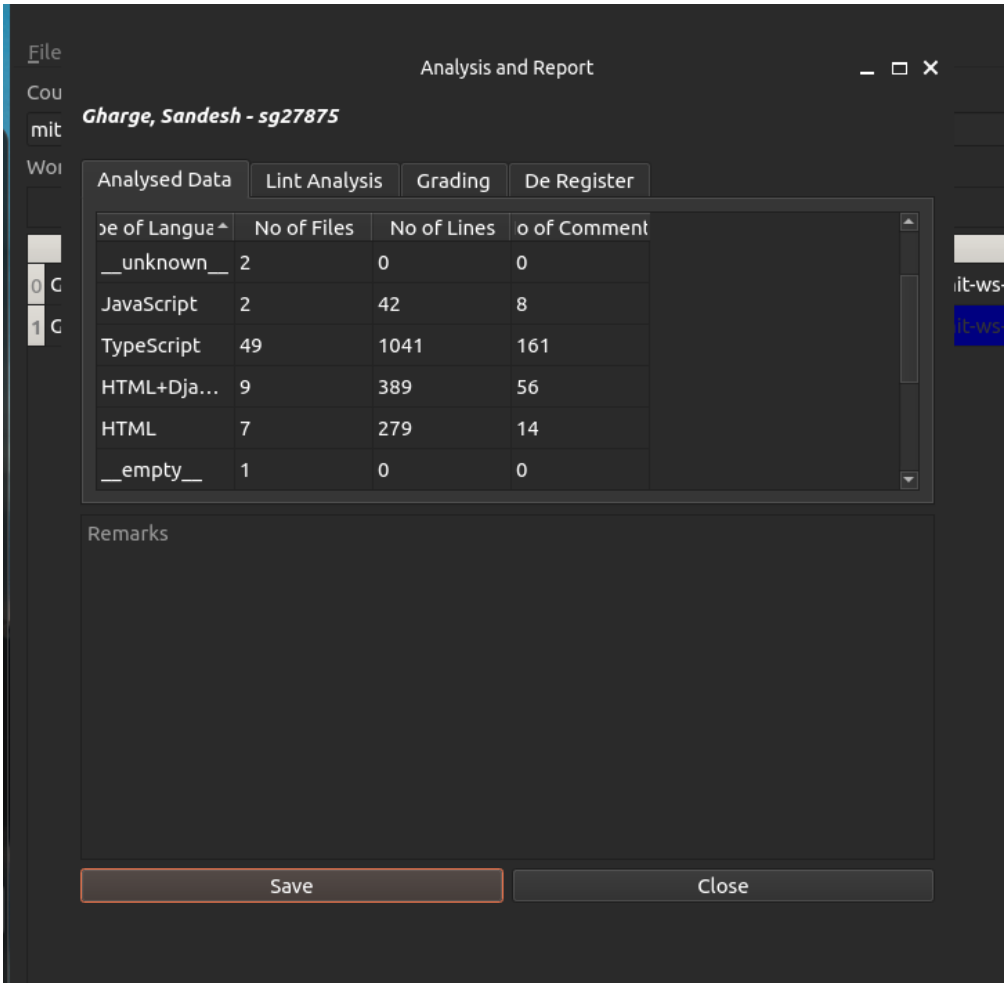

Figure 4.8: Analysis Window

The entire window is divided into 3 tabs namely - Analysed Data, Grading, De Register. Each tab it configured with a task which can assist professors to carry out their task swiftly. They are as follows -

Analysed Data

This tab is the default tab that opens when Analysis Window opens, and displays all the information of the project related files, number of lines, number of comments and much more. This

data gives an overview for analysing the project submitted by student. There are details that any professor has to know regarding the code e.g. number of line of codes or comments that can be automatically calculated using predefined tools in few seconds rather than calculating manually in every file.

Lint Analysis

This tab displays analysis of code files analysed using different Lint libraries. Data displayed lists down the missing coding standards, compilation errors (if present), issues that can arise with the time etc. all this data is followed by the Lint score out of 10 to denote the quality of code.

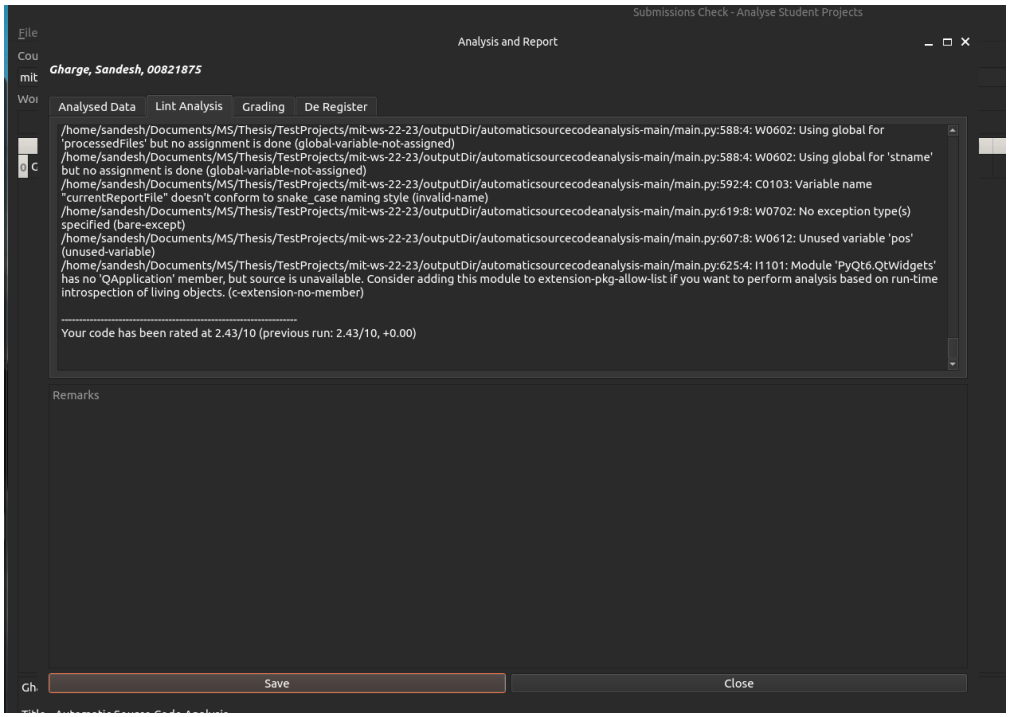

Figure 4.9: Lint Analysis Tab

Grading

Grading tab is an action tab for the professor where the submitted project can be graded. Currently there are 3 identified categories which can be used to grade a project. Referring to Fig. 4.10, they are namely ”Code”, ”Documentation”, ”Comments” added. Each category can be rated from 1 to 5 as a default value using drop down list.

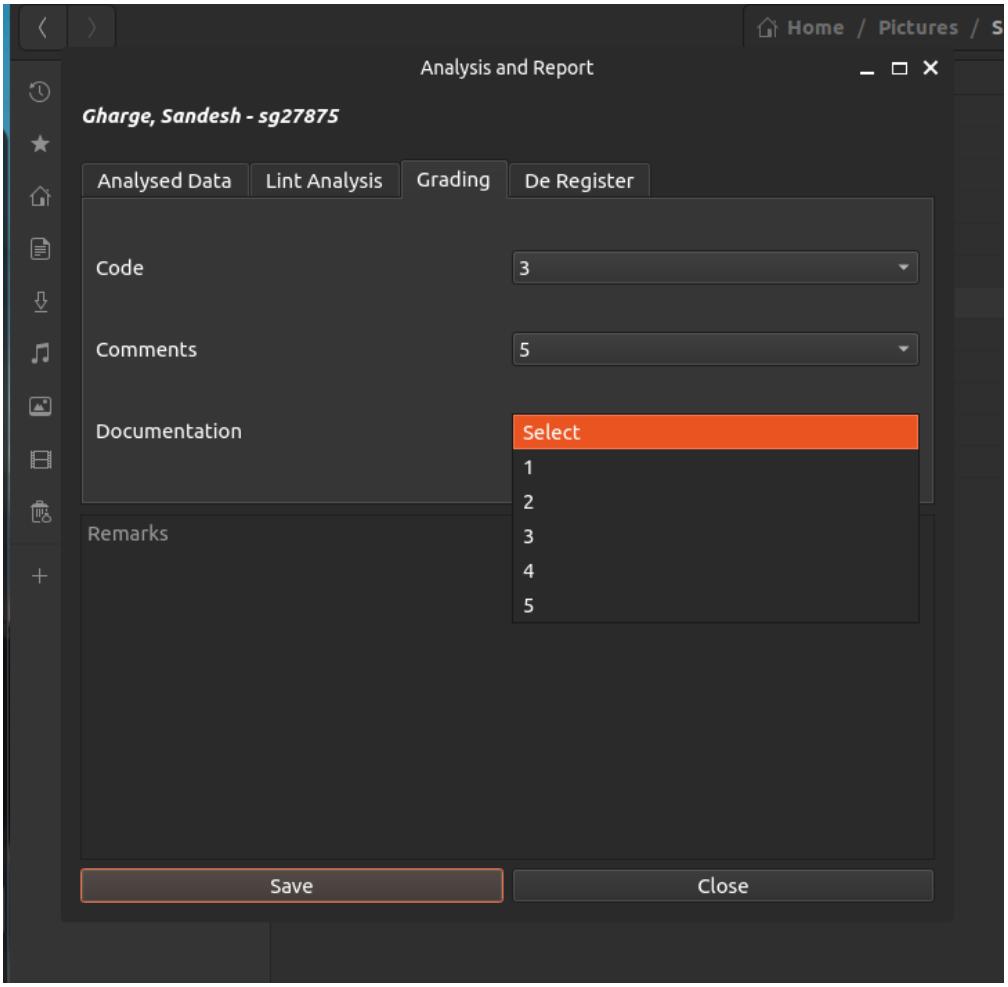

Figure 4.10: Grading Tab

Now this grading cannot be generic for all the types of project i.e., a project without any documentation, can be graded using categories Code and Comments where there is no need of documentation. Hence in such cases, the average grade should not be affected. As a result, even if the list of categories are added in the later stages, professor has to grade only those categories which are relevant and ignore others. The categories which are ignored will not be considered for average grade.

De Register

We discuss about the need to de-register from a project once evaluated in the section - 2.2.3. This problem can be solved using the last tab of Analysis Window where on click of a button, professor is de-registered from the project. Here User id and Project id are not mandatory in case the git link is available. In absence of git link, these fields can be used to de register the user from a project.

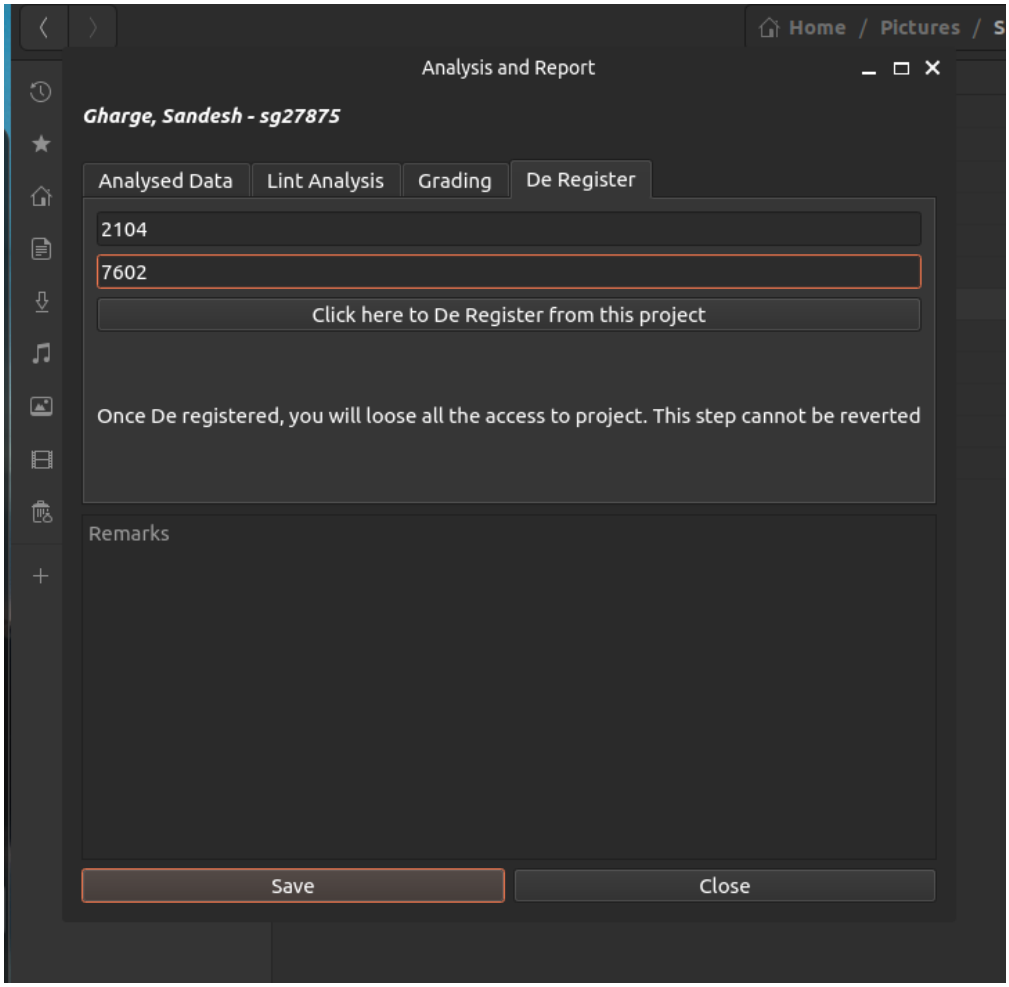

Figure 4.11: De Register Tab

Action Buttons

Once window is open professor would work on the evaluation of the project. Post evaluation, there should be a way to Save the data or close the window to start with another evaluation. Hence, there are two actions button are added at the end of the window namely ”Save” and ”Close”. On click of ”Save” button, the data is saved and passed on to the main window e.g. calculated average grade. In case, one wants to close the window for any reason, it can be done clicking on the ”Close” button.

4.2 Implementation of Design using PyQt6

After designing the main window and analysis window along and defining expected working of each window along with interaction between them, it is now time to use PyQt6 features

for implementation of the same. As discussed in the background 3 PyQt6 is rich with features which can be used to implement all the planned features of the software. Most of the classes that we are going to use for practical implementation using PyQt6 package are defined under QtWidgets [3].

4.2.1 Layout Overview PyQt6

We will see usage of many components for designing the proposed windows and arranging all components at required position can be an overhead task. Then there are layouts which are predefined in PyQt6 which helps in arranging components vertically, horizontally or even in Grid Layout. Apart from traditional layouts there are also layouts developed to give smooth user experience e.g. Stacked layout, form layout. Combination of multiple layouts are termed as Compound layouts in PyQt6. Before, developing the component it becomes necessary to know about the different types of layout which can help rapid implementation proposed design. Of all the layouts QLayout [4] is the parent layouts, inherited by QBoxLayout [5] , QGridLayout [6] , QFormLayout [7] , and QStackedLayout [8]. Below is the list layouts with its introduction-

1. QBoxLayout – It divides the area (provided by parent layout or parent) into rows or columns of boxes and adds widget in each box. This arrangement can be performed vertically or horizontally using child classes QHBoxLayout [9] and QVBoxLayout [10] respectively.
    a) QHBoxLayout – This class lines up widgets horizontally.
    b) QVBoxLayout – This class lines up widgets vertically.

2. QGridLayout – It divides the area (provided by parent layout or parent) into rows and columns of boxes and adds widget in each box. If compared to previous layout, grid layout requires 2 parameters as an address in a 2D array to add a given widget. This layout gives a better control to arrange components both vertically and horizontally.

3. QFormLayout – As the name denotes, this layout is used to create forms. When we plan to design any input field, we tend to create a text edit field and a label which gives the description to what should be the input. QFormLayout is a convenience layout class that lays out its children in a two-column form. The left column consists of labels and the right column consists of "field" widgets (line editors, spin boxes, etc.).

4. QStackedLayout – This layout allows developer to add multiple layouts together but display one at a time. It is equivalent to multiple pages stacked together and only the first page is visible.

Layout introduction will provide guidance for widget arrangement after we are ready with all the widgets. Upcoming section will implement the proposed design using PyQt6 predefined classes.

4.2.2 Main Window

Our main class that defines main window is child class to QMainWindow [11] which is defined in QtWidgets. QMainWindow provides all the features that are necessary to design a main window such as menu bar, tool bar, status bar etc. This software needs a design of one menu item in menu bar. When it comes to adding a Menu item in a menu bar below listed classes are used to create a complete the flow -

1. QMainWindow.menuBar() – Returns a object of menu bar where we can add items and buttons as required.
2. QMainWindow.menuBar().addMenu() – It is used to add a menu in the list of all the menus. It needs an argument as name of string to be displayed preceded by an '&'.
3. QAction [12] – It is used to create a button inside a menu and passed as an argument while adding action button into a given menu. To create a QAction object the constructor used in this software accepts QIcon [13], name of the action preceded by '&'.
4. QIcon – Used to create an icon object and accepts path of the image as parameter.
5. QKeySequence [14] – QAction button can be loaded with a shortcut using QKeySequence.

Refer below code snippet for understanding -
main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
        # Create a button
        imagePath = ""
        button_load = QAction(QIcon(imagePath, "&Load iLearn
        ZIP", self)

        ## setStatusTip shows tips about the action button on
        hover
        button_load.setStatusTip("Load ZIP with submission data")

        button_load.setShortcut(QKeySequence("Ctrl+o"))
        button_load.triggered.connect(self.loadiLearnZIP)

        # Add the created button into new menu item
        menu = self.menuBar()
        file_menu = menu.addMenu("&File")
        file_menu.addAction(button_load)
```

.
.
.

This way we can add as many menu item as we want with each menu item consisting QAction buttons as required.

Course Name and Working Directory

After we add the menu item and its required action buttons, we added course name using QLabel [15]. This QLabel is also used to display the Window Directory label in main window. QLabel displays text using function "setText('Label Name')", which cannot be changed with User Interference but can be updated as needed in the back end. The text field which accepts the actual value of course is defined using QLineEdit [16] class. Below code demonstrate how the QLabel and QLineEdit for above cases are implemented -

main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
        # Label for Course Name
        self.labCourseName = QLabel(self)
        self.labCourseName.setText("Course Name (used for output dir): ")

        self.leCourseName = QLineEdit(self)
        self.leCourseName.setText(courseName)

        self.courseNameLayout = QHBoxLayout(self)
        self.courseNameLayout.addWidget(self.labCourseName)
        self.courseNameLayout.addWidget(self.leCourseName)

        # Label for Working directory
        dirPath=""
        self.labWorkingDir = QLabel(self)
        self.labWorkingDir.setText("Working Directory: " + dirPath)
.
.
.
```

Access Token

Access has a similar structure to Course Name, except the field works like a password. Hence the text entered should not be visible which is achieved by using setting QLineEdit echo mode

option to password. Placeholder is used to inform the necessity of the field.
main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
        # Token label and field
         self.labToken = QLabel(self)
         self.labToken.setText("Gitlab Access Token: ")

         self.leToken = QLineEdit(self)
         self.leToken.setPlaceholderText("Enter Gitlab Token.
        (Necessary only if internet is available for project deregistration)")
         self.leToken.setEchoMode(QLineEdit.EchoMode.Password)

         self.tokenLayout = QHBoxLayout(self)
         self.tokenLayout.addWidget(self.labToken)
         self.tokenLayout.addWidget(self.leToken)
         # Token label and field end
.
.
.
```

Project Details Table

Referring to 4.1.1, where we planned to display all the details of the project in the form of table. This requirement is implemented using QTableView [17]. It needs an input in the form of model, where we have used QSortFilterProxyModel [18] enable the sorting and filtering on the data passed between another model and a view. A custom table model is passed to QSortFilterProxyModel defining the design of the table view and data is passed in the form of DataFrame from the pandas library.
main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
    # Declaring table view variable and making it Non Editable
    self.table = QtWidgets.QTableView()
    self.table.setEditTriggers(QtWidgets.QAbstractItemView.EditT
    igger.NoEditTriggers)
```

```
        # function call when click action is performed on the table
        self.table.clicked.connect(self.tabClicked)

        # function call when double click action is performed on the table to
        self.table.doubleClicked.connect(self.openAnalyserWindow)

        # Declaring data frame model where studentData is a global
        variable and columns defines the head of the column
        self.df = pd.DataFrame(studentData, columns = ['Name',
        'Grades', 'File', 'Link1', 'Link2', 'Link3'])

        # Declaring the custom model
        self.model = TableModel(self.df)

        # Finally declaring model and passging it to table view for
        display purpose
        self.proxyModel = QSortFilterProxyModel()
        self.proxyModel.setSourceModel(self.model)
        ## Setting boolean to true to enable sorting
        self.table.setSortingEnabled(True)
        self.table.setModel(self.proxyModel)
.
.
.
```

Display Panel

As discussed earlier, display panel is used to display the readme file when clicked on the readme file path on the project details table. If referred to previous code snippet, there is click function defined which checks for data if the path to readme file, if the data is displayed on the display panel. This panel is implemented using QTextEdit [19], it is similar to QLineEdit but has ability to alter multiple lines or one can say paragraph.

main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
        # Declaring a variable for display panel
        self.teProject = QTextEdit()

        # Function called on click to display readme file data on the
        display panel
```

```
    def tabClicked(self, item):
        global dataDir, currWorkingDir, outDir

        cd = item.data()
        print(cd)

        ## if cell content is a full path to a file that exists
        if (os.path.isfile(cd)):
            # open it
            with open( cd, 'r', encoding="utf-8") as f:
                print('Showing content of MD file:')
                # markdown = Markdown(extensions=['codehilite'])
                text = f.read()
                # Settting the Readme file text
                self.teProject.setText(text)
        else:
            if cd.startswith("https:"):
                print("Downloading " + cd)
                path  = currWorkingDir + os.sep + outDir

                # print("Current Working Dir: " + cwd)
                clone = "git clone " + cd # "git clone
                gitolite@<server_ip>:/your/project/name.git"

                ################# Specifying the path where the cloned project
                ################# needs to be copied
                ################# os.chdir(path)

                ################# # to-do: check whether project was already
                ################# downloaded
                ################# os.system(clone) # Cloning

                ############ else:
                ################# # otherwise show cell content
                ################# self.teProject.setText(str(cd))
.
.
.
```

Similarly, if this click happens on the cell which contains git repository link, the function called on click is also configured to clone the project on the local. Last else condition is used by rest of the cases and displays the data in its original form.

Layout Implementation

Design itself shows that the widget are arranged vertically, hence we will be using QVBoxLayout to display the required components. Although, this implementation will not allow the user to change the size of Project display table or display panel. Hence, these 2 widget are clubbed together in a QSplitter [20]. A splitter allows user to control size of the child widgets by dragging the boundary between them. Any number of widgets may be controlled by a single splitter. Below code snippet completes the main window design. Please refer to previous code snippet for variable names used -

main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
    # Declaring splitter
    splitter = QSplitter(Qt.Orientation.Vertical)

    # Adding widgets to splitter
    splitter.addWidget(frTableData)

    # Project Display table
    splitter.addWidget(self.table)

    # Display panel
    splitter.addWidget(self.teProject)

    # defining default stretch factor
    splitter.setStretchFactor(1, 1)

    # Defining a layout variable
    layout = QVBoxLayout()

    ## adding widgets in sequential order
    layout.addLayout(self.courseNameLayout)
    layout.addLayout(self.tokenLayout)
    layout.addWidget(self.labWorkingDir)
    layout.addWidget(splitter)

    cWidget = QWidget()
    cWidget.setLayout(layout)

    self.setCentralWidget(cWidget)
.
```

.
.

A vertical layout is ready, but it won't be displayed until this layout is assigned to central widget. Last three statements creates a widget which consists of layout designed and later assigned to central widget. This completes the architecture at code level to display the main window as a whole.

4.2.3 Analysis Window

Analysis window is defined in a different class with QDialog [21] as the parent class. QDialog is a base class for dialog window generally used for short term task and brief communications with the user. There are two types of dialog window defined in the PyQt6, Modal Dialogs and Modeless Dialogs.

1. Modal Dialogs
   It is a dialog which blocks input to other visible windows in the same application, e.g. a dialog used to fetch file from the local system. Here user cannot work on the parent window till any file is selected for required purpose or until the window is closed. In other words, after an application window dialog is opened, user must finish interaction with the dialog box and close before accessing other window in the application. Window model dialog blocks only those windows which are associated with the model dialog, allowing user to continue working on other windows of the application.

2. Modeless Dialogs
   This dialog operates independently of other windows in the same application. Unlike modal dialog, it doesn't block other windows.

Current analyser window is implemented using model dialog, considering a professor would work on only one project at a time. Hence, access to main window is not allowed once a analyser window is opened for a particular project.

main.py

```
class MainWindow(QtWidgets.QMainWindow):
.
.
.
    def openAnalyserWindow(self, item):
        index = item.row()
        self.analyserWindow = AnalyserWindow(lstProjData[index], index)

        # function called to save the report
        self.analyserWindow.closed.connect(self.saveReport)

        # self.analyserWindow.show() -- will open a modeless dialog
```

```
            self.analyserWindow.exec()
            return
.
.
.
```

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    def __init__(self, data, index):

        super(AnalyserWindow, self).__init__()
        .
        .
        .
        self.initUI()
        return
.
.
.
```

This completes the initialisation process of analyser window, which lead to using PyQt6 components to implement proposed design of tabs, buttons and other features.

Student Name

A student name is fetched from the project data details that appears in the first column of the project detail's table and displayed at the top of the window. This name acts as an identifier as to which project professor is evaluating. QLabel is used to display this name.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    # studObj is a student object passed
    self.projectName = QLabel(self)
    self.projectName.setText(self.stuObj.studentName)
.
.
.
```

Analysed Data

Analysed data displays all the information regarding the project files, their count, types along with lines of code, comments etc. This data is displayed in tabular form using the same custom class of Table Model using QTableView similar to 4.2.2Project Details Table. This data is calculated using tool pygount, which we will be discussing inside back end implementation.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    self.df = ... data to be populated ...

    self.tbl = TableModel(self.df)
    self.dispData = QTableView()
    self.dispData.verticalHeader().hide()
    self.dispData.setColumnWidth(0, 500)
    self.dispData.resize(200, 400)

    self.proxyModel = QSortFilterProxyModel()
    self.proxyModel.setSourceModel(self.tbl)
    self.dispData.setSortingEnabled(True)
    self.dispData.setModel(self.proxyModel)
.
.
.
```

Lint Analysis

Fourth tab is Lint Analysis that shows the analysis computed by Lint module. Currently only python files are analysed using pylint library. Data is displayed using QTextEdit widget. Lint provides detailed analysis of the code with suggestion of refactoring the code with preferred format as per coding standards. All the analysis is followed by the score out of 10. This will help professor analyse the project and grade the submission accordingly. analyseFiles function is defined to analyse the code and display the data on this tab. Detailed description of this function is explained in the back end implementation in upcoming chapter.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
```

```
        .
        .
        .
        self.la = QTextEdit(self)
        self.la.setPlaceholderText("Analysis")
        self.analyseFiles()
        self.la.setText(self.lintData)
        self.lalayout = QHBoxLayout()
        self.lalayout.addWidget(self.la)
        .
        .
        .
.
.
.
```

Grading

For grading we need drop down list to select a grade as per the evaluation, which is implemented using QComboBox [22] with default value as 'Select' and grades ranging from 1 to 5. QComboBox provides a list of item to the user for selection consuming minimum space. This combo box can be editable, meaning more values can be added whenever required but since the range is already defined between 1 to 5, editable option of Each combo box labeled with QLabel. Now there are 6 components i.e. 3 labels and 3 corresponding combo box which has to be arranged is a tabular manner so that each label will be followed by corresponding combo box in the same line. Imagine a table with 2 columns and 3 rows, with left column loaded with labels and right column with combo box as needed in the Fig. 4.10 can be arranged using QGridLayout [6]. Simple combination of indexes to address the cell in a table are used to arrange all the elements as required.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
        self.cat1 = QLabel()
        self.cat1.setText("Code")

        self.cat2 = QLabel()
        self.cat2.setText("Comments")

        self.cat3 = QLabel()
        self.cat3.setText("Documentation")
```

```
        self.cbcat1 = QComboBox()
        self.cbcat1.addItems(["Select","1","2","3","4","5"])
        self.cbcat1.setCurrentIndex(self.stuObj.grades.code)

        self.cbcat2 = QComboBox()
        self.cbcat2.addItems(["Select","1","2","3","4","5"])
        self.cbcat2.setCurrentIndex(self.stuObj.grades.comments)


        self.cbcat3 = QComboBox()
        self.cbcat3.addItems(["Select","1","2","3","4","5"])
        self.cbcat3.setCurrentIndex(self.stuObj.grades.documentation)

        self.catLayout = QGridLayout(self)
        self.catLayout.addWidget(self.cat1, 0, 0)
        self.catLayout.addWidget(self.cat2, 1, 0)
        self.catLayout.addWidget(self.cat3, 2, 0)

        self.catLayout.addWidget(self.cbcat1, 0, 1)
        self.catLayout.addWidget(self.cbcat2, 1, 1)
        self.catLayout.addWidget(self.cbcat3, 2, 1)
.
.
.
```

Remark

Since grades are not the only thing that professor might have to note down, hence a QTextEdit is used to add any remarks for reference of the professor.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    self.remarks = QTextEdit(self)
    self.remarks.setPlaceholderText("Remarks")

    # When the window opens for the same project after
    evaluation, remarks are retained
    self.remarks.setText(self.stuObj.remark)
.
```

.
.

De Register

This tab has a button to de-register from a project, so that once the professor has evaluated the project it won't visible on the dashboard of Gitlab leaving only relevant projects that are not evaluated. There are two fields that can be used to de registered a user from a project using User id and Project id. Button is implemented using QPushButton [23] with a warning message displayed using QLabel. This warning message is to notify user that once the de registration is completed, the project will not be accessible on Gitlab. These elements are arranged using vertical box layout class QVBoxLayout as shown in the code snippet below.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    # De Register Window

    self.deRegL = QVBoxLayout()

    self.userId = QLineEdit(self)
    self.userId.setPlaceholderText("Enter User ID")

    self.projId = QLineEdit(self)
    self.projId.setPlaceholderText("Enter Proj ID")

    self.deReg = QPushButton(self)
    self.deReg.setText("Click here to De Register from this project")
    self.deReg.clicked.connect(self.deRegister)
    self.deRegWarning = QLabel()
    self.deRegWarning.setText("Once De registered, you will loose all th
        access to project. This step cannot be reverted")

    self.deRegL.addWidget(self.userId)
    self.deRegL.addWidget(self.projId)
    self.deRegL.addWidget(self.deReg)
    self.deRegL.addWidget(self.deRegWarning)

    # De Register Window end
.
.
```

.

Tab Implementation

All the components, Analysed Data, Grading, De-Register and Lint Analysis windows widgets needs to arranged in the tab format. QTabWidget [24] is a widget used which allows developer to arrange multiple widgets or layouts in its own tab and access them simultaneously on click. QTabWidget also allows developer to arrange the tabs in the any direction of the window i.e. East, West, South or North (Default). But QTabWidget only accepts the widget as argument along with name of the tab in the form of string. Hence widgets variables are declared for all the tab, then assigned to respective layouts, before initialising the tab structure as below -

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    ## Tab Widget initialisation
    self.tab = QTabWidget()
    self.analysedData = QWidget()
    self.grades = QWidget()
    self.dReg = QWidget()
    self.lintAnalysis = QWidget()
    ## Tab Widget initialisation end

    ## Tab widget add layout

    self.analysedData.setLayout(self.analysedLayout)
    self.grades.setLayout(self.catLayout)
    self.dReg.setLayout(self.deRegL)
    self.lintAnalysis.setLayout(self.lalayout)

    self.tab.addTab(self.analysedData, "Analysed Data")
    self.tab.addTab(self.grades, "Grading")
    self.tab.addTab(self.dReg, "De Register")
    self.tab.addTab(self.lintAnalysis, "Lint Analysis")

    ## Tab widget add layout end
.
.
.
```

Action Buttons

Window needs to be closed and data needs to saved when evaluation is completed. These actions are completed using 2 buttons namely 'Save' and 'Close' and are placed at the end of the analyser window. Buttons are arranged in a layout horizontally using QHBoxLayout.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    # Action button layout

    self.save = QPushButton("Save",self)
    self.save.clicked.connect(self.saveData)
    self.closeBtn = QPushButton("Close", self)
    self.closeBtn.clicked.connect(self.closeAndEmitData)

    self.actionLayout = QHBoxLayout()
    self.actionLayout.addWidget(self.save)
    self.actionLayout.addWidget(self.closeBtn)

    # Action button end
.
.
.
```

Layout Implementation

There are 3 components that needs to arrange in the analyser window i.e. Name of the student, Tabs for evaluation, Remark text edit and actions buttons. QVBoxLayout is used to arrange all these components in the final process of designing analyser window. This layout is then assigned to the final layout of the analyser window.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    # Designing final Layout

    self.ml = QVBoxLayout()
    self.ml.addWidget(self.projectName)
```

```
        self.ml.addWidget(self.tab)
        self.ml.addWidget(self.remarks)
        self.ml.addLayout(self.actionLayout)

        # Designing final Layout end

        self.setLayout(self.ml)
.
.
.
```

This completes the design implementation of both Main Window and Analyser Window, its nothing but the outline of the planned software and actual working of the entire application together. Next chapter will explain the detailed explanation of logic and algorithm implemented for the software.

5 Back End Implementation

All the front end components are configured and in this section we will integrate logic and all necessary algorithms for all the components to interact with each other to give appropriate output. A project data structure is defined in a class and used in the software every where as a common mode of data packet. This synchronisation effortlessly handled the data movement from multiple components and windows. Understanding this structure is important for further implemented process, hence we will start with class used to represent a project uploaded in the software.

5.1 Project Data Class

Basic details of a project includes name of the student who submitted the project, list of files, list of source files, remark of the professor post evaluation and grading of the project. For grading a project there are multiple things to be considered like grades for different category, average grade, if the project is graded. Hence, a class is also defined for grading of a project whose instance is used as one of the variable in project data class.

projectObject.py

```
class ProjectData :
    def __init__(self):
        self.studentName : str
        self.files = []
        self.srcFiles = []
        self.remark = ""
        self.dirPath = ""
        self.grades : GradingFields = GradingFields()
        return

class GradingFields :
    def __init__(self):

        self.comments : int = 0
        self.code : int = 0
        self.documentation : int = 0
        self.graded : bool = False
        self.avgGrade : int = 0
        return
```

.
.
.

Understanding of this class will prove helpful in upcoming back end implementation of main window as well as analyser window.

5.2 Main Window

Main window of this application acts as dashboard of the software which will be currently blank since we have not loaded any project. The first step is to load the project and display details in 4.1.1 Project Details Table.

5.2.1 Load Zip File

Current implementation accepts the projects only in the form of .zip file. To load a project, click on "Load Zip file" which will open a windows dialog to select a project file of type .zip. On selection, zip is unzipped inside a folder named after value entered inside 4.1.1Course Name on the dashboard. Directory in which the files are extracted is updated in the widget of 4.1.1Working directory. After successful extraction the details are displayed on the project details table.

main.py

```
class MainWindow(QtWidgets.QMainWindow):

.
.
.
    def loadiLearnZIP(self):
        global courseName, currWorkingDir, dataDir, outDir,
        zipFileName, studentData, projData

        zipFileName = QFileDialog.getOpenFileName(self,
        'Open iLearn ZIP file',
            './' + dataDir, "ZIP files (*.zip)")

        if len(zipFileName[0]) > 0:
            path = Path(zipFileName[0])
            parentpath = str(path.parent.absolute())
            print(path)
            print(parentpath)

            # courseName can be changed in TextEdit
            # so get current name now
```

```
courseName = self.leCourseName.text()

currWorkingDir = str(parentpath) + os.sep +
courseName

## Value of working directory updated
self.labWorkingDir.setText("Working Directory: " +
currWorkingDir)

# unzip Moodle assignment ZIP
shutil.unpack_archive(zipFileName[0], currWorkingDir
+ os.sep + outDir)
fileName = zipFileName[0].split(os.sep)[-1]
folderName = fileName.split('.')[0]
projData.dirPath = currWorkingDir + os.sep + outDir +
os.sep + folderName

# tree(currWorkingDir + os.sep + outDir)
tree(projData.dirPath)

## Once traversing is complete, project details table
is updated with the data as below
self.df = pd.DataFrame(studentData, columns =
['Name', 'Grades', 'File', 'Link1', 'Link2',
'Link3'])
self.model = TableModel(self.df)
header = self.table.horizontalHeader()
header.setSectionResizeMode(0,
QHeaderView.ResizeMode.ResizeToContents)
header.setSectionResizeMode(1,
QHeaderView.ResizeMode.ResizeToContents)
header.setSectionResizeMode(2,
QHeaderView.ResizeMode.ResizeToContents)
header.setSectionResizeMode(3,
QHeaderView.ResizeMode.ResizeToContents)
header.setSectionResizeMode(4,
QHeaderView.ResizeMode.ResizeToContents)
# for sorting
self.proxyModel = QSortFilterProxyModel()
self.proxyModel.setSourceModel(self.model)

self.table.setSortingEnabled(True)

self.table.setModel(self.proxyModel)
```

```
            self.table.update()

            return True
        return False
.
.
.
```

'tree' function is used to traverse through the folder extracted and perform operations of storing list of files and source files in the project data object.

Traverse Folders

The function of the 'tree' function is to traverse through the project folder and it do not contribute directly to the functionality of the main window, hence it is defined and accessible outside the class. function 'listdir' accepts an input as path of the directory and returns all the files and folders present inside the input path. This data of files and folders is passed to function 'walk' for traversing of the directory and generating data of all the files and folders present inside the main project directory.

main.py

```
def tree(path):
    # path = str(Path(path))
    dirs, files = listdir(path)[:2]
    # print(path)

    global zipFileName, reportfile, reportfilename,
    processedFiles, studentData, lstProjData, projData
    # create report files? Global report and student's reports!
    try:
        filename = Path(path).parent.name
        print(filename)
        reportfilename = path + os.sep + filename + ".md"
        print(reportfilename)
        with open(reportfilename, 'w') as reportfile:
            processedFiles.append(reportfilename)
            walk(path, dirs, files)
            # save processed files in reportfile
            print('Processed Files: \n')
            for name in processedFiles:
                print(name + '\n')

    except:
        print('Error while walking through dir tree path: ' + path)
```

```
        # reportfile.close()
    print(processedFiles)
    print(studentData)
    lstProjData.append(projData)
    projData = ProjectData()
```

Here 'walk' function is defined for walking through the folder structure and processing all types of files. All the project data for selected projected is stored in 'projData' object and added in the list of projects as the It is possible to load multiple projects at a time.

Locate Files

The function 'walk' is a recursive function that extracts all the files and folders inside the project directory. While extracting, every file is processed using 'processFile' function. The file type can be of multiple types depending upon the type of projects submitted, like Java, Spring Boot, Rest API or Python project or MEAN Stack application that includes JavaScript, TypeScript, HTml, CSS. As a result the type of files varies but all the source files are stored together the projData object. Apart from source code files ReadMe file (Markdown file) is processed separately as the location of the ReadMe file is needs to be displayed in the project details table.

main.py

```
def walk(root, dirs, files, prefix=''):
    global reportfile, reportfilename, processedFiles, stname

    currentReportFile = reportfile
    print('Walking␣through␣root:␣' + root, currentReportFile)

    if FILES and files:
        file_prefix = prefix + ('|' if dirs else '␣') + '␣␣␣'
        # process files in root dir
        for name in files:
            print(file_prefix + name + '␣->␣processing', currentReportFile)
            # read the file and process content
            # use the data directory
            processFile(root, name) # + dataDir + 'outputDir/'
        print(file_prefix)

    dir_prefix, walk_prefix = prefix + '+---', prefix + '|␣␣␣'
    for pos, neg, name in enumerate2(dirs):
        if neg == -1:
            dir_prefix, walk_prefix = prefix + '\\---', prefix + '␣␣␣␣'
        print(dir_prefix + name)
        path = os.path.join(root, name)
```

```
        # print(name)
        # students names only from dirs on top level in Moodle ZIP
        if prefix == "":
            getStudentNameFromDirName(name)
            print(stname)
        try:
            dirs, files = listdir(path)[:2]
        except:
            pass
        else:
            walk(path, dirs, files, walk_prefix)

def processFile(dirname, name):

    print('Processing file ' + name)
    # handle files, if not already processed
    fullname = dirname + os.sep + name
    if fullname not in processedFiles:
        processedFiles.append(fullname)
        handleFileType(dirname, name)

def handleFileType(dn, fn):
    ext = ""
    # check for . in filename, so it has an extension, take last
    extension
    if '.' in fn:
        ext = fn.split('.')[-1].lower()

    if ext == 'zip':
        try:
            # unzip_recursively(dn + fn)
            # extractDir = os.path.abspath('.')

            # dfn = dn + os.sep + fn
            shutil.unpack_archive(fn, dn) # , 'zip'
            # name of unpacked dirs unknown here
            # tree(dn)
            # process the dir again
            try:
                dirs, files = listdir(dn)[:2]
            except:
                pass
            else:
                walk(dn, dirs, files, '+---')
```

```
                # good visual prefix? Must not be ''
        except:
            print('Error while trying to unzip file: ' + fn)
    if ext == 'md':
        handleMarkdown(dn, fn)
    if ext == 'pdf':
        handlePDF(dn, fn)
    if not ext:
        print('Processed file has no or unknown extension: ' + fn)
    if ext in srcFilesTypes :
        projData.srcFiles.append(fn)
    else:
        projData.files.append(fn)
```

Along with the location of the files, student name and project directory is populated in the 'projData' object. This object is then passed to Analyser Window for further process of evaluation.

5.2.2 Pass Data to Analyser Window

As we know there can be multiple projects, the project data is stored in the form of list. In the fig.4.6 multiple rows can be displayed for each project. We learnt about the single click functionality that when clicked any of the row, data is displayed in the Display Panel. To open a analyser window, we are supposed to double click the row. While opening the window, all the project details corresponding to the click are passed to the analyser window and the access token is populate into the project object in case for de-registration operation. While opening window, we use exec() function which makes the dialog window Modal dialog. It can be converted into Modeless dialog be calling show() function instead of exec().

main.py

```
class MainWindow(QtWidgets.QMainWindow):

.
.
.

    self.table.doubleClicked.connect(self.openAnalyserWindow)
.
.
.
    def openAnalyserWindow(self, item):
        index = item.row()
        lstProjData[index].accessToken = self.leToken.text()
        self.analyserWindow = AnalyserWindow(lstProjData[index], index)
```

```
        self.analyserWindow.closed.connect(self.saveReport)
        self.analyserWindow.exec()
        return
.
.
.
```

5.3 Analyser Window

This window opens with few operation before loading the entire window. pygount tool is used to analyse the project folder structure which returns all the statistics of the submitted project. It includes total number of files, number of lines of code, total number of lines, number of comments etc. for all the files types. This data is then displayed on the analysed data tab which gives an overview of the project that professor can consider for evaluation.

The tool pygount can be used as a command line tool hence cannot be integrated directly with the python code. The command is executed and response is fetched using python library in the form of JSON. This JSON object is converted into data frame object to populate the table (refer 4.2.3) as required as below -

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.

    analysedData = json.loads(subprocess.run(shlex.split('pygount
␣␣␣␣--format=json␣' + self.stuObj.dirPath), capture_output=True).stdout.d
        dispData = []
        for lang in analysedData['languages']:
            dispData.append(
                [
                    lang['language'],
                    lang['fileCount'],
                    lang['sourceCount'],
                    lang['documentationCount']
                ]
            )
        self.df = pd.DataFrame(dispData,
            columns= ['Type␣of␣Language', 'No␣of␣Files', 'No␣of
␣␣␣␣␣␣␣␣␣␣␣␣Lines', 'No␣of␣Comments']
        )

        self.tbl = TableModel(self.df)
```

```
        self.dispData = QTableView()
        self.dispData.verticalHeader().hide()
        self.dispData.setColumnWidth(0, 500)
        self.dispData.resize(200, 400)

        self.proxyModel = QSortFilterProxyModel()
        self.proxyModel.setSourceModel(self.tbl)
        self.dispData.setSortingEnabled(True)
        self.dispData.setModel(self.proxyModel)

        self.remarks = QTextEdit(self)
        self.remarks.setPlaceholderText("Remarks")
        self.remarks.setText(self.stuObj.remark)

        self.analysedLayout = QVBoxLayout(self)
        self.analysedLayout.addWidget(self.dispData)
.
.
.
```

Lint Analysis

pylint is similar to the pygount which runs on the command line and similar imlementatiion can be seen here. Only difference in the response of two libraries is pylint provides the response in the form of string unlike pygount which provides data in the form of JSON. Referring to the section 4.2.3 implementation of the function is explained in detail. The 'if' condition is present to analyse the files of type javascript and 'else' condition is analyses all the python files. Data gathered for all the files is initialised in the QTextEdit widget on the Lint Analysis window.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    .
    .
    .
    def analyseFiles(self):
        self.lintData = ""
        for file in self.stuObj.srcFiles:
            ext = file.split('.')[-1].lower()
            print(file)
            if ext == 'js' and False:
```

```
                with open(file, 'r') as f:
                    js_code = f.read()
                    print(jslint.check_JSLint(js_code))
            elif ext == 'py':
                d = subprocess.run(shlex.split('pylint '+ file), capture_
                self.lintData = self.lintData + ' \n ' + d
        return
    .
    .
    .
.
.
.
```

5.3.1 Remarks and Average Grade

Professor grades the project as per different categories and adds remarks depending upon the data shown on the analysed data window and any other factors depending upon the professor. The evaluation is saved in the project data object when 'Save' button is clicked. During the process average grade is also calculated and updated into project data object. When clicked on 'Close' button the window is closed and project data object is passed to main window using custom signal emit functionality. The data passed also includes the index of the project that is evaluated.

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    def saveData(self):
        self.stuObj.grades.code = self.cbcat1.currentIndex()
        self.stuObj.grades.comments = self.cbcat2.currentIndex()
        self.stuObj.grades.documentation = self.cbcat3.currentIndex()
        self.stuObj.grades.calAvgGrade()

        print(self.stuObj.grades.avgGrade)

        self.stuObj.remark = self.remarks.toPlainText()
        return

    def closeAndEmitData(self):
        self.closed.emit(self.stuObj, self.i)
        super().accept()
```

```
        return
.
.
.
```

projectObject.py

```
class GradingFields :
.
.
.
    def calAvgGrade(self):
        totalGrades = 0
        count = 0
        self.avgGrade = 0
        if self.comments > 0:
            count += 1
            totalGrades += self.comments

        if self.code > 0:
            count += 1
            totalGrades += self.code

        if self.documentation > 0:
            count += 1
            totalGrades += self.documentation
        if count == 0:
            self.avgGrade = 0
        else:
            self.avgGrade = totalGrades / count
.
.
.
```

5.3.2 De Registration

This is only part in the software which needs the internet connection to call the Gitlab API to de register the user from a project. This API call needs two parameters user id of the user and project id of project than has to be de registered for the user. A message pops up on execution of the process stating the result of call. API is executed using request library of the pyhton as below -

analyserWindow.py

```
class AnalyserWindow(QDialog):
.
.
.
    def deRegister(self):

        uId = self.userId.text()
        pId = self.projId.text()
        deReg_Url = GIT_LAB_URL+"/projects/"+ pId +"/members/" + uId
        print(deReg_Url)

        headers = {
            "PRIVATE-TOKEN": self.stuObj.accessToken,
            "Content-Type": "application/json"
            # You can adjust this header based on your API requirements
        }

        response = requests.delete(deReg_Url, headers=headers)
        print(response.status_code)

        if response.status_code == 204:
            QMessageBox.information(self, "De Registration Status",
            "The project has been successfully de registered.")

        elif response.status_code == 404:
           resData = json.loads(response.text)
           QMessageBox.critical(self,
           "De Registration Status", resData["message"])

        return
.
.
.
```

5.3.3 Update Grade on Main Window

The signal emitted with the project details object by Analyser window executes a triggers slot function which receives the data. This data is then updated in the main window e.g. Grade on project details table is updated with the average grade calculated in the previous section. Table is not directly updated rather new dataframe object is created and passed to the QTableView widget and entire data is refreshed. As a result we can see the grades on the main window.

main.py

```
class MainWindow(QtWidgets.QMainWindow):

.
.
.

    def openAnalyserWindow(self, item):
        index = item.row()
        self.analyserWindow = AnalyserWindow(lstProjData[index], index)
        self.analyserWindow.closed.connect(self.saveReport)
        self.analyserWindow.exec()
        return

    def saveReport(self,data : ProjectData, i):
        lstProjData[i] = data
        studentData[i][1] = data.grades.avgGrade
        self.df = pd.DataFrame(studentData, columns = ['Name', 'Grades',
        'File', 'Link1', 'Link2', 'Link3'])
        self.model = TableModel(self.df)
        header = self.table.horizontalHeader()
        header.setSectionResizeMode(0,
        QHeaderView.ResizeMode.ResizeToContents) # .Stretch to
        Maximum
        header.setSectionResizeMode(1,
        QHeaderView.ResizeMode.ResizeToContents)
        header.setSectionResizeMode(2, QHeaderView.ResizeMode.Stretch)
        # for sorting
        self.proxyModel = QSortFilterProxyModel()
        self.proxyModel.setSourceModel(self.model)

        self.table.setSortingEnabled(True)

        self.table.setModel(self.proxyModel)
        self.table.update()
        return
.
.
.
```

5.4 Save Report

After evaluation if the window is closed, all the data is lost. To save the evaluation, the project details table can be saved in the .xlsx or .csv format. There are two ways to save the data -

1. Click on File button in the menu bar and then click on the second option of Save Report
2. Click on top-right corner close button of the window that leads to pop message containing the save option

This file is saved in the folder with name corresponding to the course name given on the main window. This report can be used for further evaluation of the submissions.

main.py

```
class MainWindow(QtWidgets.QMainWindow):

.
.
.
    def saveOverviewReport(self):
        """DataFrame.to_csv(path_or_buf=None, sep=',', na_rep='',
        float_format=None, columns=None, header=True, index=True,
        index_label=None, mode='w', encoding=None,
        compression='infer', quoting=None, quotechar='"',
        lineterminator=None,
        chunksize=None, date_format=None, doublequote=True, escapechar=No
        decimal='.',
        errors='strict', storage_options=None)"""
        global currWorkingDir, courseName

        if (os.path.exists(currWorkingDir)):
            filepath = Path(currWorkingDir + os.sep
            + courseName + "-report.csv")
            # create parent dir if it does not exist
            # filepath.parent.mkdir(parents=True, exist_ok=True)
            self.df.to_csv(filepath, ";")
            filepath = Path(currWorkingDir + os.sep + courseName +
            "-report.xlsx")
            self.df.to_excel(filepath, ";") # , engine using default
            openpyxl, engine='xlsxwriter'
            return True
        else:
            pass # continue
.
.
.
```

6 Generate Executable Files

Currently, the software runs when the main window file (i.e. main.py) is executed using python in the command line.

- Command - python3 main.py

Our aim was to run the software on the desktop with different environments of operating systems. This is not how we will be running the software practically. pyinstaller module is used to generate executable files for operating systems.

- Command - pyinstaller –onefile main.py

Executable file will be generated under a dist folder. Important thing to note is the executable file will only be generated for the operating system on which the above is executed. As a result, to create the executable file for major operating system i.e. Windows, MacOS, Linux the command has to be executed on every operating system generating corresponding executable file.

7 Agile Methodology

End product of this thesis is designing an application depending upon the requirements gathered after a detailed research to minimise the efforts of professor for evaluation of the project. Every subsequent step was depending the outcome of the previous task. Hence, implementing Scrum framework was not a suitable option in this case. Also, major part of the thesis was to gather the data of requirements through research and implementing Extreme Programming would prove a wrong choice as this type has the major focus on quality of code and application. Kanban which emphasizes on continuous flow of was used as a Agile Framework to track the progress of thesis.

The flow commenced with research topics regarding the mode of submissions, general methods followed by professors or evaluators to maintain the submissions every semester, listing down the obstacles faced by professors and evaluators during evaluation of submissions and requirements of an application that prove useful in determining the platform of development that can aid the general process followed for evaluation. This research guided the flow of the thesis towards successful development of the application. All the task were documented in an excel sheet with its completion dates as below, which will give an brief overview over the progress that was made during the period.

Table 7.1: Kanban Report

Sr No	Task	To Do	In Progress	Done
1	Research general mode of submission in any University	10/05/23	11/05/23	24/05/23
2	Research how the submitted projects are handled by professors or evaluators	10/05/23	25/05/23	31/05/23
3	Research on general practice followed by professor for evaluation of the project	10/05/23	01/06/23	10/06/23
4	Detailing about the requirements which can help develop a software to aid the evaluation process of application developed	13/06/23	13/06/23	16/06/23
5	Research about type of application that can be suitable for the application development	14/06/23	19/06/23	22/06/23
6	Research about selection of framework for development of desktop software	14/06/23	23/06/23	26/06/23
7	Research about the PyQt6 Framework and its features that can be used to develop a software	26/06/23	26/06/23	28/07/23
8	Understanding the existing version of the software prepared by professor	26/06/23	27/06/23	03/08/23

Sr No	Task	To Do	In Progress	Done
9	Implementation of pop window – Analyzer Window	03/08/23	04/08/23	09/08/23
10	Implementation of Analyzed Data Tab	03/08/23	10/08/23	11/08/23
11	Implementation of Grading Tab	03/08/23	14/08/23	15/08/23
12	Implementation of De Registration tab	03/08/23	16/08/23	17/08/23
13	Add Remark section in Analyzer Window	04/08/23	18/08/23	18/08/23
14	Add Action buttons on Analyzer Window	04/08/23	21/08/23	22/08/23
15	Add functionality to calculate average grade and save the data on click of 'Save' button	21/08/23	23/08/23	25/08/23
16	Display Analyze window on double click of row displayed on the main window	22/08/23	28/08/23	28/08/23
17	Pass the project object to Analyzer Window on double click of row displayed on main window	22/08/23	29/08/23	29/08/23
18	Create a manual Signal that is emitted on closing of Analyzer window	29/08/23	30/08/23	04/09/23
19	Pass the saved object in Analyzer Window back to Main window using manual signal that is emitted	29/08/23	05/09/23	07/09/23
20	Define a slot in main window that fetches the object emitted by Analyzer window on closer	29/08/23	07/09/23	13/09/23
21	Add a column in Main Window to display average grade calculated in Analyzer window	05/09/23	14/09/23	15/09/23
22	Define the functionality in the slot of displaying the average grade in the object fetched on the main window	05/09/23	18/09/23	25/09/23
23	Create a label at the top of Analyzer window to display student name	05/09/23	26/09/23	26/09/23
24	Fetching Student Name from ReadME File	05/09/23	27/09/23	28/09/23
25	Populating the project object with student name and displaying it on main window	05/09/23	28/09/23	29/09/23
26	Add Lint Analysis Tab in Analyzer window	20/09/23	29/09/23	29/09/23
27	Issue : ReadMe file path getting blank on double click while opening the Analyzer window	28/09/23	28/09/23	02/10/23
28	Issue : More than 1 rows are displayed for projects having multiple markdown files	29/09/23	02/10/23	03/10/23
29	Analyze python files using pylint	20/09/23	04/10/23	07/10/23
30	Issue : Double click not working for all projects on loading of more than 1 projects	07/10/23	09/10/23	12/10/23
31	Issue : Open Analyzer Window as Modal Dialog	07/10/23	20/10/23	25/10/23
32	Issue : Data not transferred on converting Analyzer Window into Modal dialog	25/10/23	25/10/23	26/10/23
33	Issue : Data not updated on main window after the data passed from Analyzer window to main window	25/10/23	25/10/23	27/10/23

Sr No	Task	To Do	In Progress	Done
34	Display analyzed data on Lint Analysis tab	20/09/23	08/10/23	27/10/23
35	Add Editable fields to fetch user id and project id to de register from the project	27/10/23	27/10/23	28/10/23
36	Complete the functionality of de registration calling the GitLab API	27/10/23	28/10/23	29/10/23
37	Re arrange the Analyzer Tabs in proper sequence	31/10/23	01/11/23	03/11/23
38	Create a field to enter Access Token	03/11/23	04/11/23	05/11/23
39	Pass the access token while opening the Analysis Window	04/11/23	04/11/23	05/11/23

8 User Guide

This section will provide a guidance to end users on how the software works and what are things that has to be taken care of before to start the work on submission analysis.This guidance will explain the steps to follow so that one can use this software in best possible way. Below are the steps below for guidance-

1. A project that has to be analysed needs to have only one ReadMe.md file in entire workspace. Multiple ReadMe files can cause the application to work with inaccuracy.
2. The ReadMe file should have the name of the student in the first line.
3. After the project structure is ready, open the application.
4. When the main window opens for the first time the dashboard will be blank.
5. Edit the course name field. The text entered will be used as a folder name to extract all the folders that will be loaded in to the application for evaluation.
6. Working Directory field will tell the exact location of the project being extracted.
7. Click on the option 'Load iLearn ZIP' in the Menu Bar under files which will open a windows dialog to select a .zip file of the project that has to be analysed.
8. Once the .zip file is selected, a row is displayed on the main window with the data of Name of the student, path to ReadMe file, link to git repository and other links if available.
9. On clicking the cell with ReadMe path, the file content is displayed in the display panel below.
10. On clicking other cells, the text displayed in the cell is displayed as it is in the display panel.
11. On double clicking anywhere on the row, will open an analysis window.
12. Analysis window has student name at the top of the window and middle section comprises of multiple tabs followed by the remark section and at the end action buttons which includes 'Save' and 'Close'.
13. This analysis window will have the default tab open with the name Analysed Window.
14. Analysed window will have all the data summary regarding all the files present in the loaded project.

15. Grading Tab is where you can grade the submission.
16. Lint Analysis tab shows the detailed analysis of all the files present in the project.
17. De-Registration Tab is present for de-registering from a project whose evaluation has been completed. This way the user is de-registered as a member of the project repository in the Gitlab. Note that for this process to happen system must be connected to internet for API call. Also, user must know its user id and project id for the process to get complete.
18. For de-registration process, user needs to complete access token configuration. i.e.
    a) Login into Gitlab account
    b) Go to settings
    c) Click on Access Token configuration
    d) Generate a token giving all the access to the token
    e) Configure this token inside on the main window the software
    f) Now with the internet connection you can enter the user id and project id and click on De-Registration button, This will remove the user with user id from the project with project id entered.
19. The remark section is provided to add any remarks regarding the analysis of the project.
20. On Completing all the operations, user can click on save button and close the window. This action will update the main window with average grade considering the grades provided in the analyses window.
21. Now on main window the average grade is visible on the dashboard corresponding to the project that has been evaluated.
22. To save the evaluation, select the 'Save Overview Report' option from Menu Bar under File option.
23. This will complete one flow of evaluating a project. The details are saved in the form of .csv or .xlsx format in the same folder where .zip folder has been extracted.

9 Results

The designed software is integrated with features that can aid the evaluation process carried by an evaluator or professor by extracting details of projects automatically. Simple UI interface makes it easy to use and save the report as needed. Analysed window display data regarding file structure i.e. total number of files, total number of source files, total number of each type of programming file, in addition number of lines of code, number of blank lines, number of lines of comments. Using Lint Analysis the result can even check if the files submitted are compiled without errors. This way, evaluator can have a good idea before running the submitted application for testing.

Regarding de-registration process the planned implementation stated, use of git clone link to de-register from the project. It was not possible to implement hence, as a workaround, user can de-register from the project using user id and project id with current version of the software. This way practically it is even possible to de-register from other projects using current project window. Apart from this functionality all other target functionality is developed and tested.

9.1 Limitations

The product developed is a first version of the desktop software providing all the basic features for reviewing a project developed in any platform. Though this software executes its task swiftly and efficiently, it comes with limitations which can be rectified to develop a new version of software which can be useful for vast range of users. We will talk about few of them in details -

9.1.1 Limited Programming Language support for Analysis

The version developed during the course of this thesis supports Analysis using Lint on project based on Python or JavaScript using pylint and pyjslint. Hence analysis is confined to narrow range of projects and this can be solved if the support is diversified for wide range of languages which will lead to analysing the projects based on broad range of platforms.

9.1.2 Dependency on ReadMe File

Information displayed on the main window mainly depends on the ReadMe mark down file which is one of the common file present in almost every project. This software traverse all files and projects to get a list of source files, mark down files and other files. This markdown file is used to extract the information related to project i.e. Name of the student, link to the git repository and other links that are associated with the project. Absence or blank ReadMe

file in the project will not be a suitable for Code review and analysis using this software. This software depends on the structure of the readme file which is mandatory.

9.1.3 Selection of Single Project at a time

With current implementation, a user can only select one .zip at a time, there is no support if multiple project .zip files are selected at the same time. In case multiple .zip files are selected, only the first file is considered for analysis, hence to load other projects same number of click operation has to be performed.

9.1.4 De Registration not possible offline

The de-registration process in the software depends on the API call to Gitlab. This will be not possible if the system do not have internet connection.

10 Planned Features of the software

A brief inspection of limitation can suggest numerous improvements a software can have in the upcoming versions of the software. Here are a few points which can prove useful in upgrading the software with necessary features to improve the quality and reliability of the designed tool.

10.1 Lint Analysis Support to Other multple languages

Current version of the software supports Lint analysis of source files in Python and JavaScript. This support can be extended to analysis of popular and well established programming languages making the software to work with wide range of projects.

10.2 Automatic Grading using Lint Analysis

Lint analysis provides detailed report about the errors on code, coding standards followed, possible problems with the structure of the code etc. This report is used by the professor to manually grade the submissions. Using Data Analytic and Machine Learning, it will be possible to analyse the report generated by Lint and grade the submission automatically. In this way, with expert programmers in the field of Artificial Intelligence and Machine Learning, grading can be made automatic leading to creation of an application with minimal involvement of professors or evaluators. This can be used specially for platforms that provides online education through courses with videos like Moocs. It will eliminate the delay caused by manual evaluation and instant report can help a learner to continue without any obstacle.

10.3 Absolute Software

The designed software is loaded with basic task of extracting summary of an application developed, platform to add grades and save the data necessary for evaluation of the submissions in any University. As a desktop software there are many basic features missing in current version of the software, e.g. In Menu bar, File option is loaded with recently accessed projects, Preference settings dealing with the looks to software i.e. Dark Mode or Light mode, Font to be used, Size of the Font as per users needs and many more. Though the current version will prove useful to the professors at personal level, it can be upgraded with common features that a desktop software should possess leading to development of an absolute software. This software then can be preferred by many developers, learners for reviewing purpose.

10.4 Tool support for multiple languages

The software designed has interface language as English. Though, English is globally excepted language few user can prefer the language they are comfortable with. Incorporating few of these languages can prove useful for wide range of potential users.

11 Conclusion

When it comes to curriculum, students are putting efforts to reach the pace of growth of technologies used for application development and mentors are continuously trying to bring down the gap of real world technology and the syllabus followed in the every University. In one of the research we saw a gradual advancement in mode of submissions that was followed by the student in Universities. While handling all these submissions, we addressed numerous challenges faced by majority of the professors across the globe. Considering all these factors, we designed a software with an attempt to reduce the workload of professors or in general, evaluators which can significantly save the time that can be utilised for other useful purposes.

Software was designed considering the requirements listed after a brief discussion with Professor Udo Garmann and brain storming the common issues that any professor might face during their tasks. Before the start of development, a detailed research on selection of platform i.e. Web Development vs Desktop Software Development was performed. Desktop Software Development was preferred choice considering all the requirements that were listed down which mainly focuses on the application that can work offline and has no collaboration with other professors. This research was followed by selection of framework for development of the software. Major challenge in development of desktop software is compatibility of the code with multiple platforms or in other words operating systems. Python based framework PyQt6 was the preferred option out of many for its popularity in multiple domains such as Data Analysis, Artificial Intelligence or Scientific Computing and many more. Availability of tons of libraries helps developer with rapid development and easy maintenance of the code. This selection also allowed to maintain single code base as PyQt6 framework allows cross-compatibility i.e. a single code base can be used to run the software in majority of the operating systems like Windows, MacOS, Linux.

When it comes to research question *What are the key challenges faced by professors and academic managers when handling a significant number of student projects, and how can technology mitigate these challenges effectively?*, this software has significantly reduced the overhead for professor with the task such as getting details of files (number of files, type of files, etc.) and measure of code written and comments using the Analysed Data tab in the Analyser Window. With the grading tab and remarks text field, professor can maintain the records in a file for a particular course in a semester. With minimal efforts of changing the course in the Main Window, professor can start working on different course whose results will be maintained in a new folder.

Evaluation of details related to files was not enough, hence use of Static Code Analysis was introduced with the question - *Can Static code analysis help professor while evaluation of the projects?* Analysing code submitted for errors can be a tedious task which can be achieved using Static Code Analysis. Lint was preferred choice of option as a Static Code Analyzer for its suitable features as compared to other static code analyzers. Lint is light weight compared

to some static code analyzers and fast in terms of feedback which makes it well-suited for integration into the development process without causing significant delays. It imposes coding standards which results into consistent code formatting and naming conventions. These standards improves readability and easy collaboration within a team which aids in easy maintenance. It can also be used as a part of automation in integral part of CI/CD pipeline to keep check on the code base. One of the feature of Lint, though not used in this project can prove useful in upcoming versions, i.e. customization where you can allow Lint to adapt to specific coding standards according to the projects.

To conclude, this a first version of software which is capable of doing basic tasks with ease and upcoming versions can prove prominent.

Bibliography

[1] Ivo Gomes, Pedro Morgado, Tiago Gomes, Rodrigo Moreira, "An overview on the Static Code Analysis approach in Software Development." [Online]. Available: https://paginas.fe.up.pt/~ei05021/TQSO%20-%20An%20overview%20on%20the%20Static%20Code%20Analysis%20approach%20in%20Software%20Development.pdf

[2] Darko Stefanović, Danilo Nikolić, Dušanka Dakić, Ivana Spasojević & Sonja Ristić, "STATIC CODE ANALYSIS TOOLS: A SYSTEMATIC LITERATURE REVIEW." [Online]. Available: https://www.daaam.info/Downloads/Pdfs/proceedings/proceedings_2020/078.pdf

[3] PyQt documentation, "QWidget." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QWidget.html

[4] ——, "QLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QLayout.html

[5] ——, "QBoxLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QBoxLayout.html

[6] ——, "QGridLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QGridLayout.html

[7] ——, "QFormLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QFormLayout.html

[8] ——, "QStackedLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QStackedLayout.html

[9] ——, "QHBoxLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QHBoxLayout.html

[10] ——, "QVBoxLayout." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QVBoxLayout.html

[11] ——, "QMainWindow." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QMainWindow.html

[12] ——, "QAction." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QAction.html

[13] ——, "QIcon." [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtGui/QIcon.html

[14] ——, “QKeySequence.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtGui/QKeySequence.html

[15] ——, “QLabel.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QLabel.html

[16] ——, “QLineEdit.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QLineEdit.html

[17] ——, “QTableView.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QTableView.html

[18] ——, “QSortFilterProxyModel.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtCore/QSortFilterProxyModel.html

[19] ——, “QTextEdit.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QTextEdit.html

[20] ——, “QSplitter.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QSplitter.html

[21] ——, “QDialog.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QDialog.html

[22] ——, “QComboBox.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QComboBox.html

[23] ——, “QPushButton.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QPushButton.html

[24] ——, “QTabWidget.” [Online]. Available: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QTabWidget.html