**МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ “ЛЬВІВСЬКА ПОЛІТЕХНІКА”**

**КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# **КУРСОВА РОБОТА**

з дисципліни «Об'єктно-орієнтоване програмування»

на тему: «Модель навчання в приватній школі»

студента групи ПЗ-21

спеціальності 121

«Інженерія програмного забезпечення»

Каська Р.В.

Керівник: доцент кафедри ПЗ,

к.ф.н., доцент Дяконюк Л. М.

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

Львів — 2020

**Зміст**

Завдання ……………..…………………………………………………………………….3

Алгоритми розв’язку задачі ……………………………………………………..……….3

Діаграма класів ……………………………………………………………………………8

Діаграма прецедентів ……………………………………………………………………..9

Діаграма послідовності виконання ……………………………………………………..10

Код програми …………………………………………………………………………….13

Протокол роботи програми для кожного пункту завдання…………….……………..40

Інструкція користувача та системні вимоги ……………………………………………46

Опис виняткових ситуацій ………………………………………………………………54

Висновки ……………………………………………………...………………………….65

Список використаної літератури ……………………………………………………….66

1. **Завдання**

Створити програму, що відтворює модель навчання в приватній школі. У закладі є перелік викладачів та дисциплін, які вони викладають. Відомі прізвища викладачів, їх посада та стаж роботи.

Студент, що має прізвище, номер курсу та групи, в якій навчається, та перелік дисциплін, відповідно до курсу. Він має мати змогу вибрати викладача, що може викладати цю дисципліну.

Потрібно створити наступні сервіси:

1. Наповнення інформацією про викладачів та дисципліни.
2. Можливість вибору викладачів за дисциплінами.
3. Можливість перегляду для викладачів студентів курсу та всіх студентів, що записалися за курсами викладача.
4. Визначення найпопулярнішого викладача за заданою дисципліною.
5. Сортування списку викладачів за посадою та прізвищем.
6. Визначити всіх викладачів, що мають тільки один курс.
7. Пошук не зайнятих спеціалістів на задану дату.
8. Автоматичне очищення історії від застарілих дат.
9. **Алгоритм розв‘язку задачі у покроковому представленні**

Розробив покроковий алгоритм роботи програми загалом та покрокові алгоритми кожного пункту функціоналу.

* 1. Даний алгоритм наповнює інформацією про викладачів та дисципліни. Покроковий опис алгоритму F:

F1. Створення об’єкту класу QFile та передача шляху до файлу зі списком даних.

F2. Відкриття файлу.

F3. Перевірка вмісту файлу на коректність даних.

F4. Якщо помилку знайдено, то вивід повідомлення про неї та закриття файлу.

F5. Відкриття файлу.

F6. Зчитування інформації по рядку та зберігання її у системі.

F7. Закриття файлу.

* 1. Даний алгоритм дає можливість вибору викладачів за дисциплінами. Покроковий опис алгоритму C:

C1. Вибір дисципліни зі QComboBox, що містить дисципліни відповідно до курсу студента.

C2. Вибір елемента – «Show My disc Teacher List». Він виводить на QTableWidget список викладачів, які мають вибрану дисципліну.

C3. Вибір із таблиці QTableWidget викладача, та подвійний клік на його прізвище. Елемент появиться навпроти дисципліни студента.

C4. Вибір елемента – «Choose teacher on this discipline». Він зчитує прізвище з поля, де відображені дисципліни і викладачі студента, перевіряє коректність інформації, та записує у змінну IdOfPersons викладачу ID даного студента, а відповідно студенту ID викладача. Коли студент обрав собі викладачів під кожен предмет, з’явиться information QMessageBox із текстом всередині: «Successful added your teachers».

* 1. Даний алгоритм дає можливість перегляду для викладачів студентів курсу та всіх студентів, що записалися за курсами викладача.

Покроковий опис алгоритму SL:

SL1. Цикл із індексом проходження k = 0..discipline.size(). Виконувати кроки SL2 – SL4.

SL2. Цикл із індексом проходження i = 0..studentList.size(). Виконувати кроки SL3 – SL4.

SL3. Цикл із індексом проходження j = 0..stidentList[i].GetSizeOfIdOfPersons(). Виконувати крок SL4.

SL4. Перевірка чи даний ID викладача є у списку ID викладачів у студента. Якщо так, то виконати крок SL5. Якщо ми шукаємо викладачів курсу, то ще умови чи у даній дисципліні є наш викладач.

SL5. Додавання у атрибут викладача QVector<QPair<QString,QString>> enrolledStudentsInfo прізвище студента, та його групу.

* 1. Даний алгоритм визначає найпопулярнішого викладача за заданою дисципліною.

Покроковий опис алгоритму MP:

MP1. Перевірка чи передана дисципліна коректна.

MP2. Цикл із індеком проходження i = 0..teacherList.size(). Виконувати кроки MP3-MP4.

MP3. Цикл із індеком проходження j = 0..teacherList[i].GetDisciplines().size(). Виконувати крок MP4.

MP4. Якщо даний викладач містить цю дисципліну, запис його у вектор teacherVect викладачів.

MP5. Ініціалізація змінної max, у якій зберігається кількість записаних учнів у першого викладача.

MP6. Цикл із індеком проходження i = 1..teacherVect.size(). Виконати крок MP7.

MP7. Перевірка чи max < teacherVect[i].GetSizeOfIdOfPersons(). Якщо справджується умова, то запис у max нового значення кількості записаних студентів.

MP8. Повернення max та запис інформації про найпопулярнішого викладача.

* 1. Даний алгоритм сортує список викладачів за посадою та прізвищем.

Покроковий опис алгоритму SP:

SP1. Перевантаження оператора <.

SP2. Виклик функції std::sort, та передача їй таких параметрів: teachersList.begin(), teacherList.end(), лямбда функція [](Teacher &firstTeacherSurname, Teacher &secondTeacherSurname) {*return* firstTeacherSurname < secondTeacherSurname;}).

Аналогічний алгоритм для сортування посад.

* 1. Даний алгоритм визначає всіх викладачів, що мають тільки один курс. Покроковий опис алгоритму OC:

OC1. Цикл із індексом проходження i = 0..teacherList.size(). Виконувати крок OC2.

OC2. Перевірка чи даний викладач (teacherList[i]) є тільки один предмет (.GetDiscipline().size()). Якщо умова правильна, то запис його у вектор викладачів.

* 1. Даний алгоритм виконує пошук не зайнятих спеціалістів на задану дату.

Покроковий опис алгоритму FT:

FT1. Цикл з індексом проходження i = 0...teacherList.size(). Виконувати кроки FT2-FT7.

FT2. Ініціалізація bool flag = true.

FT3. Цикл з індексом проходження j = 0..teacherList[i].GetDisciplines().size(). Виконувати кроки FT4-FT6.

FT4. Зчитування інфформації про дисципліну, а саме її початковий термін, кінцевий термін та дні викладання.

FT5. Цикл з індексом проходження k = 0..disciplineDays().size(). Виконувати кроки FT6.

FT6. Перевірка чи сьогоднішня дата міститься у межах між початковий терміном та кінцевим терміном. Якщо умова справджується, то flag = false. та виконання кроку FT5.

FT7. Перевірка чи flag = true. Якщо умова справджується, то запис у вектор викладачів даного.

* 1. Даний алгоритм виконує автоматичне очищення історії від застарілих дат. Покроковий опис алгоритму CD:

CD1. Цикл з ітератором it = listOfDisciplines.begin()..listOfDisciplines.end(). Виконувати CD2-CD5.

CD2. Цикл за індексом проходження i = 0..teacherList.size(). Виконувати пункт CD3-CD5.

CD3. Перевірка чи у переданій дисципліні кінцевий термін є ранішим ніж сьогоднішня дата. Якщо умова ствердна, та виконуємо пункт CD4.

CD4. Цикл за індексом проходження k = 0..teacherList.size(). Виконувати пункт CD5-CD6.

CD5. Цикл за індексом проходження j = 0..teacherList[k].GetDisciplines().size(). Виконувати пункт CD6.

CD6. Перевірка чи дана застаріла дисципліна є у викладача. Якщо умова ствердна, то видаляєм дану дисципліну у викладача.

CD7. Цикл з ітератором iterator = listOfDisciplines.begin()..listOfDiscipline.end(). Виконувати пункт CD8.

CD8. Перевірка чи дана дисципліна є застарілою. Якщо умова виконується, то видаляєм даний предмет зі списку.

1. **UML діаграма  класів**



1. **UML діаграма  прецедентів**



1. **UML діаграми  послідовності виконання**

**5.1 Діаграма послідовності виконання (користувач-студент)**



**5.2 Діаграма послідовності виконання (користувач-викладач)**



**5.3 Діаграма послідовності виконання (користувач-адміністратор)**



1. **Код розробленої програми з коментарями**

**~~~~~~~~discipline.h~~~~~~~~**

#ifndef DISCIPLINE\_H

#define DISCIPLINE\_H

#include <QMainWindow>

#include <QDate>

#define DISCIPLINE\_DAYS\_START\_INDEX 4

*//* *клас* *Предмет,* *у* *якого* *є* *ім'я,* *курс,* *початковий* *термін* *та* *кінцевий,*

*//* *і* *дні* *тижня,* *коли* *його* *викладають*

*class* **Discipline**

{

*public*:

**Discipline**(){};

void **SetCourse**(int);

void **SetName**(QString);

void **SetStartDate**(QString date);

void **SetEndDate**(QString date);

void **SetDaysInWeek**(QVector<QString> days);

void **SetDayInWeek**(QString day);

int **GetCourse**();

QString **GetName**();

QString **GetStartDate**();

QString **GetEndDate**();

QVector<QString> **GetDaysInWeek**();

QString **GetDayByIndex**(unsigned int index);

int **GetCountOfDays**();

*private*:

int course;

QString name;

QString startDate;

QString endDate;

QVector<QString> daysInWeek;

};

*//* *глобальний* *список* *предметів*

*extern* QList<Discipline> listOfDisciplines;

#endif *//* *DISCIPLINE\_H*

**~~~~~~disciplineMethods.cpp~~~~~**

#include «discipline.h»

void Discipline::**SetCourse**(int course)

{

*this*->course = course;

}

void Discipline::**SetName**(QString name)

{

*this*->name = name;

}

void Discipline::**SetStartDate**(Qstring date)

{

*this*->startDate = date;

}

void Discipline::**SetEndDate**(Qstring date)

{

*this*->endDate = date;

}

void Discipline::**SetDaysInWeek**(Qvector<Qstring> days)

{

*this*->daysInWeek = days;

}

void Discipline::**SetDayInWeek**(Qstring day)

{

*this*->daysInWeek.push\_back(day);

}

int Discipline::**GetCourse**()

{

*return* course;

}

Qstring Discipline::**GetName**()

{

*return* name;

}

Qstring Discipline::**GetStartDate**()

{

*return* startDate;

}

Qstring Discipline::**GetEndDate**()

{

*return* endDate;

}

Qvector<Qstring> Discipline::**GetDaysInWeek**()

{

*return* daysInWeek;

}

Qstring Discipline::**GetDayByIndex**(unsigned int index)

{

*if*(index > daysInWeek.size())

*return* «Error»;

*return* daysInWeek[index];

}

int Discipline::**GetCountOfDays**()

{

*return* daysInWeek.size();

}

**~~~~~~disciplinemanager.h~~~~~**

#ifndef DISCIPLINEMANAGER\_H

#define DISCIPLINEMANAGER\_H

#include "discipline.h"

*//Клас,* *що* *працює* *із* *списком* *дисциплін*

*class* **DisciplineManager**

{

*public*:

**DisciplineManager**();

int **FindCourseOfSubject**(QString subject);

QVector<QString> **GetDaysOfDiscipline**(QString disciplineName);

QString **GetStartDate**(QString disciplineName);

QString **GetEndDate**(QString disciplineName);

bool **IsOutdatedDiscipline**(QList<Discipline>::iterator it);

bool **IsOutdatedDiscipline**(Discipline lastElem);

void **DeleteAllInfoAboutOutdatedSubjects**();

void **DeleteOutdatedDisciplinesFromList**();

void **DeleteOutdatedDisciplinesInfoInTeachers**();

int **DeleteDisciplineInfoInTeacher**(QList<Discipline>::iterator it);

};

#endif *//* *DISCIPLINEMANAGER\_H*

**~~~~~~disciplinemanager.cpp~~~~~**

#include "disciplinemanager.h"

#include "Teacher.h"

DisciplineManager::**DisciplineManager**(){}

*//Знаходить* *відповідну* *дисципліну* *зі* *списку* *за* *ім'ям* *і* *повертає* *її* *курс*

int DisciplineManager::**FindCourseOfSubject**(QString subject)

{

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(subject == listOfDisciplines[i].GetName())

*return* listOfDisciplines[i].GetCourse();

}

*//Знаходить* *відповідну* *дисципліну* *зі* *списку* *за* *ім'ям* *і* *повертає* *дні* *її* *викладання*

QVector<QString> DisciplineManager::**GetDaysOfDiscipline**(QString disciplineName)

{

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(listOfDisciplines[i].GetName() == disciplineName)

*return* listOfDisciplines[i].GetDaysInWeek();

}

*//Знаходить* *відповідну* *дисципліну* *зі* *списку* *за* *ім'ям* *і* *повертає* *початковий* *термін* *викладання*

QString DisciplineManager::**GetStartDate**(QString disciplineName)

{

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(listOfDisciplines[i].GetName() == disciplineName)

*return* listOfDisciplines[i].GetStartDate();

}

*//Знаходить* *відповідну* *дисципліну* *зі* *списку* *за* *ім'ям* *і* *повертає* *кінцевий* *термін* *викладання*

QString DisciplineManager::**GetEndDate**(QString disciplineName)

{

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(listOfDisciplines[i].GetName() == disciplineName)

*return* listOfDisciplines[i].GetEndDate();

}

*//Видаляє* *застарілі* *дисципліни* *зі* *списку* *предметів* *та* *з* *вчителів,* *що* *їх* *викладали*

void DisciplineManager::**DeleteAllInfoAboutOutdatedSubjects**()

{

*if*(listOfDisciplines.isEmpty()) *return*;

DeleteOutdatedDisciplinesInfoInTeachers();

DeleteOutdatedDisciplinesFromList();

}

*//Видаляє* *застарілі* *дисципліни* *із* *даних* *вчителів* *що* *їх* *викладали*

void DisciplineManager::**DeleteOutdatedDisciplinesInfoInTeachers**()

{

bool deleteMode;

int teacherIndex;

*for*(*auto* it = listOfDisciplines.begin(); it < listOfDisciplines.end(); ++it)

*for*(int i = 0; i < teachersList.size(); ++i)

{

deleteMode = IsOutdatedDiscipline(it);

*if*(deleteMode)

teacherIndex = DeleteDisciplineInfoInTeacher(it);

*if*(teachersList[teacherIndex].GetDisciplines().size() == 0)

teachersList.removeAt(teacherIndex);

}

}

*//Видаляє* *застарілі* *дисципліни* *зі* *списку* *дисциплін*

void DisciplineManager::**DeleteOutdatedDisciplinesFromList**()

{

QString preLastName = listOfDisciplines[listOfDisciplines.size()-2].GetName();

bool deleteMode;

*for*(*auto* it = listOfDisciplines.begin(); it < listOfDisciplines.end(); ++it)

{

QString name = it->GetName();

deleteMode = IsOutdatedDiscipline(it);

*if*(deleteMode && name == preLastName)

{

bool checkLastEl = IsOutdatedDiscipline(listOfDisciplines.last());

*if*(checkLastEl)

listOfDisciplines.pop\_back();

}

*else* *if*(deleteMode)

it = listOfDisciplines.erase(it);

}

}

*//Перевіряє* *чи* *передана* *дисципліна* *є* *застарілою,* *повертає* *true* *-* *якщо* *застаріла*

bool DisciplineManager::**IsOutdatedDiscipline**(QList<Discipline>::iterator it)

{

QDate currentDate = QDate::currentDate();

QDate cmpEndDate = QDate::fromString(it->GetEndDate(),"dd/MM/yyyy");

*if*(currentDate > cmpEndDate)

*return* *true*;

}

*//Перевіряє* *чи* *передана* *дисципліна* *є* *застарілою,* *повертає* *true* *-* *якщо* *застаріла*

bool DisciplineManager::**IsOutdatedDiscipline**(Discipline lastElem)

{

QDate currentDate = QDate::currentDate();

QDate cmpEndDate = QDate::fromString(lastElem.GetEndDate(),"dd/MM/yyyy");

*if*(currentDate > cmpEndDate)

*return* *true*;

}

*//Видаляє* *передану* *дисципліну* *у* *вчителів,* *повертає* *індекс* *вчителя* *у* *котрого* *видалили* *дисципліну*

int DisciplineManager::**DeleteDisciplineInfoInTeacher**(QList<Discipline>::iterator it)

{

*for*(int i = 0; i < teachersList.size(); ++i)

{

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

{

*if*(teachersList[i].GetDisciplines()[j].GetName() == it->GetName())

{

*if*( j == teachersList[i].GetDisciplines().size()-1)

{

teachersList[i].GetDisciplines().pop\_back();

*return* i;

}

*else* *if*( j == 0 )

{

teachersList[i].GetDisciplines().pop\_front();

*return* i;

}

*else*

{

teachersList[i].GetDisciplines().removeAt(j);

*return* i;

}

}

}

}

}

**~~~~~~~file.h~~~~~~~**

#ifndef FILE\_H

#define FILE\_H

#include <QMainWindow>

#include <QFile>

#include <QFileDialog>

*//Клас,* *який* *працює* *із* *файлами.*

*class* File

{

*public*:

File();

QString path;

void OpenFileWithChoise(QMainWindow \*window);

void ReadAndCheckFilesInfo(QMainWindow \*window);

void ReadFromFileDisciplines(QString path);

void ReadFromFileTeachers(QString path);

void ReadFromFileStudents(QString path);

void WriteToFileDisciplines(QString path, QMainWindow \*window);

void WriteToFileNewDiscipline(QString path, QMainWindow \*window);

void WriteToFileTeachers(QString path, QMainWindow \*window);

void WriteToFileStudents(QString path, QMainWindow \*window);

void CheckIfOpenFile(QFile &file, QString name);

void CheckIfEmptyFile(QFile &file, QString name);

void CheckReadFileLinesIfEmpty(QFile &file, QString name);

void CheckStudentReadFileFormat(QString path);

void CheckTeacherReadFileFormat(QString path);

void CheckDisciplineReadFileFormat(QString path);

};

#endif *//* *FILE\_H*

**~~~~~fileMethods~~~~~~**

#include "file.h"

#include "discipline.h"

#include "disciplinemanager.h"

#include "Teacher.h"

#include "student.h"

#include <QMessageBox>

#define COURSES 4

File::File(){}

*//* *Функція,* *у* *якій* *зчитується* *інформація* *зі* *всіх* *необхідних* *файлів*

*//* *Та* *перевіряє* *правильність* *даних* *у* *файлах*

void File::ReadAndCheckFilesInfo(QMainWindow \*window)

{

*try*

{

CheckDisciplineReadFileFormat("data\\Disciplines.txt");

} *catch* (exception &ex) {

QMessageBox::critical(window,"Disciplines file read info error", ex.what());

exit(0);

}

ReadFromFileDisciplines("data\\Disciplines.txt");

*try*

{

CheckStudentReadFileFormat("data\\StudentsList.txt");

} *catch* (exception &ex) {

QMessageBox::critical(window,"Students file read info error", ex.what());

exit(0);

}

ReadFromFileStudents("data\\StudentsList.txt");

*try*

{

CheckTeacherReadFileFormat("data\\TeachersList.txt");

} *catch* (exception &ex) {

QMessageBox::critical(window,"Teachers file read info error", ex.what());

exit(0);

}

ReadFromFileTeachers("data\\TeachersList.txt");

}

*//Перевіряє* *на* *правильність* *файл* *із* *дисциплінами*

void File::CheckDisciplineReadFileFormat(QString path)

{

*//Correct* *format:* *1* *Programming* *01/09/2020* *31/05/2021* *Mon* *Wed* *Fri*

QFile file(path);

QVector<QString> days = {"Mon","Tue","Wed","Thu","Fri"};

QTextStream inDiscipline(&file);

QString lineDicsipline;

int counter = 0;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces*

*try*

{

CheckIfOpenFile(file, path); *//* *Перевірка* *чи* *успішко* *відкритий* *файл*

CheckIfEmptyFile(file,path); *//* *Перевірка* *чи* *файл* *не* *порожній*

CheckReadFileLinesIfEmpty(file,path); *//* *Перевірка* *чи* *не* *містить* *порожніх* *лінійок*

} *catch* (exception &ex)

{

*throw* runtime\_error(ex.what());

*return*;

}

file.open(QFile::ReadOnly | QFile::Text);

*while*(!inDiscipline.atEnd())

{

++counter;

lineDicsipline = inDiscipline.readLine();

QStringList query = lineDicsipline.split(rx);

Discipline lesson;

QString course = query.at(0);

*if*(course.toInt() > COURSES || course.toInt() < 1)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " course num wrong");

*for*(int i = 0; i < course.size(); ++i)

*if*(!course.at(i).isDigit()) *//* *Перевірка* *чи* *номер* *курсу* *є* *цифрою*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " course wrong");

lesson.SetName(query.at(1));

*for*(int i = 0; i < lesson.GetName().size(); ++i) *//* *Перевірка* *чи* *назви* *предметів* *є* *літерами,* *а* *якщо* *ні,* *то* *допускаються* *символи* *'-'* *та* *'\_'*

*if*(!lesson.GetName().at(i).isLetter() && lesson.GetName().at(i) != "\_" && lesson.GetName().at(i) != "-")

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " subject name wrong");

lesson.SetStartDate(query.at(2));

*for*(int i = 0; i < lesson.GetStartDate().size(); ++i) *//* *Перевірка* *чи* *початковий* *термін* *є* *у* *форматі* *дд/мм/рррр*

*if*(!lesson.GetStartDate().at(i).isDigit() && lesson.GetStartDate().at(i) != "/")

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " first date wrong. Correct format 'dd/MM/yyyy'");

lesson.SetEndDate(query.at(3));

*for*(int i = 0; i < lesson.GetEndDate().size(); ++i) *//* *Перевірка* *чи* *кінцевий* *термін* *є* *у* *форматі* *дд/мм/рррр*

*if*(!lesson.GetEndDate().at(i).isDigit() && lesson.GetEndDate().at(i) != "/")

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " second date wrong. Correct format 'dd/MM/yyyy'");

*for*(int i = DISCIPLINE\_DAYS\_START\_INDEX, disciplineDayNum = 1; i < query.size(); ++i, ++disciplineDayNum)

{

QString day = query.at(i);

*if*(day.isEmpty()) *continue*;

*for*(int j = 0; j < day.size(); ++j)

*if*(!day.at(j).isLetter()) *//* *Перевірка* *чи* *назва* *дня* *є* *словом*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() +\

" day " + QString::number(disciplineDayNum).toStdString() + " wrong");

bool check = *false*;

*for*(int i = 0; i < days.size(); ++i) *//* *Перевірка* *чи* *назви* *дня* *відповідають* *формату* *('Mon','Tue','Wed','Thu','Fri')*

*if*(day == days.at(i))

{

check = *true*;

*break*;

}

*if*(!check)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() +\

" day " + QString::number(disciplineDayNum).toStdString() + " format wrong. It should be ('Mon','Tue','Wed','Thu','Fri')");

}

}

file.close();

}

*//Зчитує* *дані* *із* *файлу* *з* *дисциплінами*

void File::ReadFromFileDisciplines(QString path)

{

QFile disciplinesFile(path);

disciplinesFile.open(QFile::ReadOnly | QFile::Text);

QString courseDisciplinesLine;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//RegEx* *for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces(\s+)*

QTextStream inputDisciplines(&disciplinesFile);

inputDisciplines.skipWhiteSpace();

QStringList disciplineInfoList;

*while*(!inputDisciplines.atEnd())

{

Discipline tmpDiscipline;

disciplineInfoList.clear();

courseDisciplinesLine = inputDisciplines.readLine(); *//Зчитує* *з* *файлу* *по* *реченню*

disciplineInfoList = courseDisciplinesLine.split(rx);*//* *Розділяє* *речення* *на* *слова*

tmpDiscipline.SetCourse(disciplineInfoList.at(0).toInt());

tmpDiscipline.SetName(disciplineInfoList.at(1));

tmpDiscipline.SetStartDate(disciplineInfoList.at(2));

tmpDiscipline.SetEndDate(disciplineInfoList.at(3));

*if*(tmpDiscipline.GetStartDate() >= tmpDiscipline.GetEndDate())

*continue*;

*for*(int j = 4; j < disciplineInfoList.size(); ++j)

*if*(!disciplineInfoList[j].isEmpty())

tmpDiscipline.SetDayInWeek(disciplineInfoList.at(j));

tmpDiscipline.SetName(tmpDiscipline.GetName().replace("\_"," "));

listOfDisciplines.push\_back(tmpDiscipline);*//* *Добавляю* *дисципліну* *у* *список*

}

disciplinesFile.close();

}

*//Перевіряє* *на* *правильність* *файл* *із* *студентами*

void File::CheckStudentReadFileFormat(QString path)

{

QFile file(path);

QTextStream inStudent(&file);

QString lineStudent;

int counter = 0;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces*

*try*

{

CheckIfOpenFile(file, path);

CheckIfEmptyFile(file,path);

CheckReadFileLinesIfEmpty(file,path);

} *catch* (exception &ex)

{

*throw* runtime\_error(ex.what());

*return*;

}

file.open(QFile::ReadOnly | QFile::Text);

*while*(!inStudent.atEnd())

{

++counter;

lineStudent = inStudent.readLine(); *//Зчитує* *з* *файлу* *по* *реченню*

QStringList query = lineStudent.split(rx); *//* *Розділяє* *речення* *на* *слова*

Student person;

person.surname = query.at(0);

*for*(int i = 0; i < person.surname.size(); ++i)

*if*(!person.surname[i].isLetter()) *//* *Перевіряє* *чи* *прізвище* *є* *словом*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " surname wrong");

QString year = query.at(1);

*if*(year.toInt() > COURSES || year.toInt() < 1)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " year wrong");

*for*(int i = 0; i < year.size(); ++i)

*if* (!year[i].isDigit()) *//* *Перевіряє* *чи* *курс* *студента* *є* *цифрою*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " year wrong");

person.SetGroup(query.at(2));

bool checkNum=*false*,checkName=*false*;

*for*(int i = 0; i < person.GetGroup().size(); ++i) *//* *Перевіряє* *чи* *група* *студента* *містить* *букви* *та* *цифри* *а* *також* *символ* *'-'*

{

*if*(person.GetGroup()[i].isDigit()) checkNum = *true*;

*if*(person.GetGroup()[i].isLetter()) checkName = *true*;

*if*(person.GetGroup()[i].isPunct() && person.GetGroup()[i] != "-")

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " group wrong");

}

*if*(!checkName || !checkNum)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " group wrong");

person.SetPersonalId(query.at(3));

*for*(int i = 0; i < person.GetPersonalId().size(); ++i)

*if* (!person.GetPersonalId()[i].isDigit()) *//* *Перевіряє* *чи* *ID* *студента* *є* *числом*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " ID wrong");

*for*(int j = DISCIPLINE\_AREA, i = 1; j < query.size(); ++j, ++i)

{

*if* (query[j] == "")

*break*;

QString teacherId = query.at(j);

*for*(int k = 0; k < teacherId.size(); ++k)

*if* (!teacherId[k].isDigit()) *//* *Перевіряє* *чи* *ID* *вчителів* *студента* *є* *числами*

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() +\

" Teacher`s " + QString::number(i).toStdString() + " ID wrong");

}

}

file.close();

}

*//Зчитує* *дані* *із* *файлу* *з* *студентами*

void File::ReadFromFileStudents(QString path)

{

QFile studentFile(path);

QString lineStudent;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//RegEx* *for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces*

studentFile.open(QFile::ReadOnly | QFile::Text);

QTextStream in(&studentFile);

in.skipWhiteSpace();

int counter = 0;

*while*(!in.atEnd())

{

lineStudent = in.readLine();

++counter;

}

studentFile.close();

studentFile.open(QFile::ReadOnly | QFile::Text);

int j = 0;

*for* (int i = 0; i < counter; ++i)

{

lineStudent = in.readLine();

QStringList query = lineStudent.split(rx);

QVector<QString> tmpVect;

Student person;

person.surname = query.at(0);

person.SetYear(query.at(1).toInt());

person.SetDisciplinesToStudent(person,person.GetYear());

person.SetGroup(query.at(2));

person.SetPersonalId(query.at(3));

person.SetPassword(query.at(4));

*for*(int j = DISCIPLINE\_AREA; j < query.size(); ++j)

{

*if*(query.at(j) == "")

*continue*;

tmpVect.push\_back(query.at(j));

}

person.SetIdOfPersons(tmpVect);

studentsList.push\_back(person); *//* *Добавляю* *студента* *до* *списку*

++j;

}

studentFile.close();

}

*//Перевіряє* *на* *правильність* *файл* *із* *вчителями*

void File::CheckTeacherReadFileFormat(QString path)

{

QFile file(path);

QTextStream inTeacher(&file);

QString lineTeacher;

int counter = 0;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces*

QRegExp surnameCheck("(\\d\*|\\W\*)");*//* *a* *digit* *(\d),* *zero* *or* *more* *times* *(\*),* *(\w\*)* *word* *character* *more* *than* *1*

QRegExp idAndExperienceCheck("\\D\*");

*try*

{

CheckIfOpenFile(file, path);

CheckReadFileLinesIfEmpty(file,path);

CheckIfEmptyFile(file,path);

} *catch* (exception &ex)

{

*throw* runtime\_error(ex.what());

*return*;

}

file.open(QFile::ReadOnly | QFile::Text);

*while*(!inTeacher.atEnd())

{

++counter;

Teacher person;

lineTeacher = inTeacher.readLine();

QStringList query = lineTeacher.split(rx);

person.surname = query.at(0);

*for*(int i = 0; i < person.surname.size(); ++i)

*if*(!person.surname[i].isLetter())

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " surname wrong");

QString experience = query.at(1);

*if*(experience < 1)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " experience wrong");

*if* (idAndExperienceCheck.exactMatch(experience))

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " experience wrong");

QPair<int,QString> tmpPos;

tmpPos.second = query.at(2);

person.SetPosition(tmpPos);

person.SetPriorityToTeacherPosition();

*if*(person.GetPosition().first < 0)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " position wrong");

QString teacherID = query.at(3);

*for*(int i = 0; i < teacherID.size(); ++i)

*if*(!teacherID[i].isDigit())

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " ID wrong");

*for*(int j = DISCIPLINE\_AREA, disciplineNum = 1; j < query.size(); ++j, ++disciplineNum)

{

*if* (query.at(j).isEmpty())

*break*;

QString checkString = query.at(j);

*for*(int z = 0; z < checkString.size(); ++z)

{

*if*(!checkString[z].isLetter() && checkString[z] != '\_' && checkString[z] != '-')

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " discipline " +\

QString::number(disciplineNum).toStdString()+ ": name format wrong");

}

bool check = *false*;

*for*(int k = 0; k < listOfDisciplines.size(); ++k)

{

QString name = listOfDisciplines[k].GetName();

QString formatName = name.replace(" ","\_");

*if*(formatName == query.at(j))

check = *true*;

}

*if*(!check)

*throw* runtime\_error("In file: " + path.toStdString() + " line " + QString::number(counter).toStdString() + " discipline " +\

QString::number(disciplineNum).toStdString()+ ": name wrong");

}

}

file.close();

}

*//Зчитує* *інформацію* *про* *вчителів* *із* *файлу*

void File::ReadFromFileTeachers(QString path)

{

QFile teacherFile(path);

QString lineTeacher;

QRegExp rx("(\\ |\\,|\\:|\\t|\\s+)"); *//RegEx* *for* *'* *'* *or* *','* *or* *'.'* *or* *':'* *or* *'\t'* *or* *spaces*

teacherFile.open(QFile::ReadOnly | QFile::Text);

QTextStream inTeacher(&teacherFile);

int counter2 = 0;

*while*(!inTeacher.atEnd())

{

lineTeacher = inTeacher.readLine();

++counter2;

}

teacherFile.close();

teacherFile.open(QFile::ReadOnly | QFile::Text);

int k = 0;

*for* (int i = 0; i < counter2; ++i)

{

int indexCounter = 0;

lineTeacher = inTeacher.readLine();

QStringList query = lineTeacher.split(rx);

Teacher person;

QString position;

QVector<QString> discipline;

person.surname = query.at(0);

person.SetExperience(query.at(1).toInt());

QPair<int,QString> tmpPos;

tmpPos.second = query.at(2);

person.SetPosition(tmpPos);

person.SetPriorityToTeacherPosition();

person.SetPersonalId(query.at(3));

person.SetPassword(query.at(4));

Discipline tmp;

QVector<QString> tmpDays;

DisciplineManager tmpManager;

*for*(int l = DISCIPLINE\_AREA, b = 0; l < query.size(); ++l, ++b)

{

tmp.SetName(query.at(l));

tmp.SetName(tmp.GetName().replace("\_"," "));

tmp.SetCourse(tmpManager.FindCourseOfSubject(tmp.GetName()));

tmp.SetStartDate(tmpManager.GetStartDate(tmp.GetName()));

tmp.SetEndDate(tmpManager.GetEndDate(tmp.GetName()));

tmp.SetDaysInWeek(tmpManager.GetDaysOfDiscipline(tmp.GetName()));

person.SetDiscipline(tmp);

}

*for*(int m = 0; m < studentsList.size(); ++m)

*for*(int l = 0; l < studentsList[k].GetSizeOfIdOfPersons(); ++l)

{

*if*(l > studentsList[m].GetSizeOfIdOfPersons()-1)

*continue*;

*if*(person.GetPersonalId() == studentsList[m].GetIdOfPersonsByIndex(l))

{

person.SetIdOfPersonsByIndex(studentsList[m].GetPersonalId(),indexCounter);

++indexCounter;

}

}

teachersList.push\_back(person); *//* *вношу* *вчителя* *у* *список*

++k;

}

teacherFile.close();

}

*//Перевіряє* *чи* *файл* *успішно* *відкрився*

void File::CheckIfOpenFile(QFile &file, QString name)

{

*if*(!file.open(QFile::ReadOnly | QFile::Text))

*throw* runtime\_error(" Could not open " + name.toStdString() + " file ");

}

*//Перевіряє* *чи* *файл* *не* *є* *порожнім*

void File::CheckIfEmptyFile(QFile &file, QString name)

{

*if*(!file.size())

*throw* runtime\_error(" File " + name.toStdString() + " is empty ");

}

*//Перевіряє* *чи* *у* *файлі* *відсутні* *порожні* *рядки*

void File::CheckReadFileLinesIfEmpty(QFile &file, QString name)

{

QTextStream inTeacher(&file);

*while*(!inTeacher.atEnd())

{

*if*(inTeacher.readLine().isEmpty())

*throw* runtime\_error("File: " + name.toStdString() + " has empty lines");

}

file.close();

}

*//Записує* *список* *вчителів* *у* *файл*

void File::WriteToFileTeachers(QString path, QMainWindow \*window)

{

QFile teacherFile(path);

*if*(!teacherFile.open(QFile::WriteOnly | QFile::Text))

{

QMessageBox::warning(window,"File","File not open");

*return*;

}

QTextStream out(&teacherFile);

int lenght = teachersList.length();

*for*(int i = 0; i < teachersList.size(); ++i)

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

teachersList[i].SetDisciplineNameByIndex(j,teachersList[i].GetDisciplines()[j].GetName().replace(" ","\_"));

*for* (int i = 0; i < lenght; ++i)

{

*//Correct* *format:* *Nickerman* *4* *Teacher* *198765* *nickerman123* *Algebra* *Geometry*

QString disciplines = "";

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

{

*if*(j == teachersList[i].GetDisciplines().size()-1)

disciplines += teachersList[i].GetDisciplines()[j].GetName();

*else*

disciplines += teachersList[i].GetDisciplines()[j].GetName() + " ";

}

QString text = teachersList[i].surname + "\t" + QString::number(teachersList[i].GetExperience()) + "\t" +\

teachersList[i].GetPosition().second + "\t" + teachersList[i].GetPersonalId() + "\t" +\

teachersList[i].GetPassword() + "\t" + disciplines + "\n";

out << text;

}

teacherFile.flush();

teacherFile.close();

}

*//Записує* *у* *файл* *нову* *дисципліну*

void File::WriteToFileNewDiscipline(QString path, QMainWindow \*window)

{

QFile disciplinesFile(path);

QString courseDisciplinesLine;

*try*

{

CheckIfOpenFile(disciplinesFile, path);

} *catch* (exception &ex)

{

QMessageBox::critical(window,"IT Private school: File error",ex.what());

*return*;

}

disciplinesFile.close();

disciplinesFile.open(QIODevice::WriteOnly | QIODevice::Append);

QTextStream out(&disciplinesFile);

QString days="";

*for*(int i = 0; i < listOfDisciplines.last().GetDaysInWeek().size(); ++i)

days += listOfDisciplines.last().GetDaysInWeek()[i] + " ";

*//Correct* *format:* *2* *Algorithms* *01/09/2020* *31/05/2021* *Mon* *Thu* *Fri*

listOfDisciplines.last().SetName(listOfDisciplines.last().GetName().replace(" ","\_"));

QString text = "\n" + QString::number(listOfDisciplines.last().GetCourse()) + "\t" + listOfDisciplines.last().GetName() + "\t" +\

listOfDisciplines.last().GetStartDate() + "\t" + listOfDisciplines.last().GetEndDate() + "\t" + days;

out << text;

disciplinesFile.flush();

disciplinesFile.close();

listOfDisciplines.last().SetName(listOfDisciplines.last().GetName().replace("\_"," "));

}

*//Записує* *список* *дисциплін* *у* *файл*

void File::WriteToFileDisciplines(QString path, QMainWindow \*window)

{

QFile disciplinesFile(path);

*try*

{

CheckIfOpenFile(disciplinesFile, path);

} *catch* (exception &ex)

{

QMessageBox::critical(window,"IT Private school: File error",ex.what());

exit(0);

}

disciplinesFile.close();

disciplinesFile.open(QFile::WriteOnly | QFile::Text);

QTextStream out(&disciplinesFile);

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

{

QString days="";

*for*(int j = 0; j < listOfDisciplines[i].GetCountOfDays(); ++j)

days += listOfDisciplines[i].GetDayByIndex(j) + " ";

QString text;

*if*(i == 0)

{

listOfDisciplines[i].SetName(listOfDisciplines[i].GetName().replace(" ","\_"));

text = QString::number(listOfDisciplines[i].GetCourse()) + " " + listOfDisciplines[i].GetName() + "\t" +\

listOfDisciplines[i].GetStartDate() + "\t" + listOfDisciplines[i].GetEndDate() + "\t" + days;

}

*else*

{

listOfDisciplines[i].SetName(listOfDisciplines[i].GetName().replace(" ","\_"));

text = "\n" + QString::number(listOfDisciplines[i].GetCourse()) + " " + listOfDisciplines[i].GetName() + "\t" +\

listOfDisciplines[i].GetStartDate() + "\t" + listOfDisciplines[i].GetEndDate() + "\t" + days;

}

*//Correct* *format:* *2* *Algorithms* *01/09/2020* *31/05/2021* *Mon* *Thu* *Fri*

out << text;

listOfDisciplines[i].SetName(listOfDisciplines[i].GetName().replace("\_"," "));

}

disciplinesFile.flush();

disciplinesFile.close();

}

*//Записує* *список* *студентів* *у* *файл*

void File::WriteToFileStudents(QString path, QMainWindow \*window)

{

QFile studentsFile(path);

*try*

{

CheckIfOpenFile(studentsFile, path);

} *catch* (exception &ex)

{

QMessageBox::critical(window,"IT Private school: File error",ex.what());

exit(0);

}

studentsFile.close();

studentsFile.open(QFile::WriteOnly | QFile::Text);

QTextStream out(&studentsFile);

*for*(int i = 0; i < studentsList.size(); ++i)

{

QString teachersIDs="";

*for*(int j = 0; j < studentsList[i].GetSizeOfIdOfPersons(); ++j)

teachersIDs += studentsList[i].GetIdOfPersonsByIndex(j) + " ";

QString text;

*if*(i == 0)

{

text = studentsList[i].surname + " " + QString::number(studentsList[i].GetYear()) + "\t" +\

studentsList[i].GetGroup() + "\t"+ studentsList[i].GetPersonalId() + "\t" +\

studentsList[i].GetPassword() + "\t" + teachersIDs;

}

*else*

{

text = "\n" + studentsList[i].surname + " " + QString::number(studentsList[i].GetYear()) + "\t" +\

studentsList[i].GetGroup() + "\t"+ studentsList[i].GetPersonalId() + "\t" +\

studentsList[i].GetPassword() + "\t" + teachersIDs;

}

*//Correct* *format:* *Savkiv* *4* *PZ-46* *400045* *31.12.2000* *460046* *346000* *200100* *100100* *999999*

out << text;

}

studentsFile.flush();

studentsFile.close();

}

*//Відкриває* *вікно* *для* *вибору* *'.txt'* *файла*

void File::OpenFileWithChoise(QMainWindow \*window)

{

*this*->path = QFileDialog::getOpenFileName(window,"Open the file", " " , "\*.txt");

}

**~~~~~~human.h~~~~~~**

#ifndef HUMAN\_H

#define HUMAN\_H

#include <QMainWindow>

#include <string>

#include <QDebug>

*using* *namespace* std;

#define NUMS\_OF\_DISCIPLINES 5

#define DISCIPLINE\_AREA 5

*//Клас,* *який* *наслідується* *вчителем* *і* *учнем.*

*class* **Human**

{

*public*:

QString surname;

**Human**(QString surname);

**Human**();

~**Human**(){};

void **SetPassword**(QString pass);

void **SetIdOfPersonsByIndex**(QString id, int index);

void **SetIdOfPersons**(QVector<QString> IDs);

void **SetPersonalId**(QString id);

QString **GetPassword**();

QString **GetIdOfPersonsByIndex**(int index);

QVector<QString> **GetIdOfPersons**();

QString **GetPersonalId**();

int **GetSizeOfIdOfPersons**();

void **ClearDataAboutIdOfPersons**();

*private*:

QString password;

QVector<QString> idOfPersons;

QString personalId;

};

#endif *//* *HUMAN\_H*

**~~~~~humanMethods.cpp~~~~~~**

#include "human.h"

Human::Human(){}

Human::Human(QString surname)

{

*this*->surname = surname;

}

void Human::SetPassword(QString pass)

{

password = pass;

};

QString Human::GetPassword()

{

*return* password;

};

void Human::SetPersonalId(QString id)

{

personalId = id;

}

QString Human::GetPersonalId()

{

*return* personalId;

}

void Human::SetIdOfPersons(QVector<QString> IDs)

{

idOfPersons.clear();

*for*(int i = 0; i < IDs.size(); ++i)

{

idOfPersons.push\_back(IDs[i]);

}

}

QVector<QString> Human::GetIdOfPersons()

{

*return* idOfPersons;

}

*//Очищає* *вектор* *із* *записаними* *ID* *людей*

void Human::ClearDataAboutIdOfPersons()

{

idOfPersons.clear();

}

*//Записує* *інформацію* *про* *ID* *людини* *у* *задану* *позицію*

void Human::SetIdOfPersonsByIndex(QString id, int index)

{

*if*(index - idOfPersons.size() == 0)

idOfPersons.push\_back(id);

*else* *if*(index - idOfPersons.size() > 1)

qDebug() << "Index is more than size()";

*else* *if*(index == 0)

idOfPersons.push\_back(id);

*else*

idOfPersons.replace(index,id);

}

QString Human::GetIdOfPersonsByIndex(int index)

{

*if*(index > idOfPersons.size())

{

qDebug() << "Index is more than size()";

*return* "Error";

}

*else* *return* idOfPersons[index];

}

int Human::GetSizeOfIdOfPersons()

{

*return* idOfPersons.size();

}

**~~~~~~~~loginpage.h~~~~~~~~**

#ifndef LOGINPAGE\_H

#define LOGINPAGE\_H

#include <QDialog>

#include <QMessageBox>

#include <QMainWindow>

*//Клас,* *який* *відповідає* *за* *вікно* *входу* *в* *систему*

*namespace* Ui {

*class* Loginpage;

}

*class* Loginpage : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* Loginpage(QWidget \*parent = *nullptr*);

void CheckLoginInput();

void CheckLoginAdmin(QString surname, QString password);

void CheckLoginTeacher(QString surname, QString password);

void CheckLoginStudent(QString surname, QString password);

void SetTeacherInfo(QString surname, QString password, int index);

void SetStudentInfo(QString surname, QString password, int index);

void SetWrongLoginMessage(QString whatsWrongString);

void ShowStudentWindow();

void ShowTeacherWindow();

void ShowAdminWindow();

~Loginpage();

*private* *slots*:

void on\_pushButton\_Login\_clicked();

void on\_commandLinkButton\_clicked();

void on\_commandLinkButton\_BackToTitle\_clicked();

*private*:

Ui::Loginpage \*ui;

};

*//Глобальні* *змінні,* *які* *відповідають* *за* *те,* *який* *користувач* *зайшов* *в* *систему(Адмін,* *Вчитель,* *Учень).*

*extern* bool CheckAdmMode,CheckStudMode,CheckTeachMode;

#endif *//* *LOGINPAGE\_H*

**~~~~~~~loginpage.cpp~~~~~~~**

#include "loginpage.h"

#include "ui\_loginpage.h"

#include "mainwindow.h"

#include "registrationpage.h"

#include "studentpage.h"

#include "teacherpage.h"

#include "student.h"

#include "Teacher.h"

Student student;

Teacher teacher;

Loginpage::Loginpage(QWidget \*parent):

QDialog(parent),

ui(*new* Ui::Loginpage)

{

ui->setupUi(*this*);

*if*(!CheckStudMode)

{

ui->commandLinkButton->hide();

ui->label\_ToRegistration->hide();

}

QRegExp rx("[a-zA-Z]+");

QValidator \*validator = *new* QRegExpValidator(rx, *this*);

ui->lineEdit\_login->setValidator(validator);

ui->label\_Info->hide();

Qt::WindowFlags flags = windowFlags();

Qt::WindowFlags closeFlag = Qt::WindowCloseButtonHint;

flags = flags & (~closeFlag);

setWindowFlags(flags);

}

Loginpage::~Loginpage()

{

*if*(CheckStudMode)

CheckStudMode = !CheckStudMode;

*delete* ui;

}

*//після* *натискання* *цієї* *кнопки* *зчитує* *інформацію* *про* *прізвище* *та* *пароль,*

*//і* *викликає* *функцію,* *що* *перевіряє* *їх* *формат*

*//та* *функції,* *які* *шукають* *відповідного* *користувача* *серед* *адміна* *та* *списків* *учнів* *і* *вчителів*

void Loginpage::on\_pushButton\_Login\_clicked()

{

CheckLoginInput();

QString login = ui->lineEdit\_login->text();

QString password = ui->lineEdit\_password->text();

CheckLoginAdmin(login,password);

CheckLoginStudent(login,password);

CheckLoginTeacher(login,password);

}

*//Перевіряє* *ввід* *прізвища* *та* *пароля*

void Loginpage::CheckLoginInput()

{

QPalette palette = ui->label\_Info->palette();

*if*(ui->lineEdit\_login->text().isEmpty())

{

palette.setColor(ui->label\_Info->backgroundRole(), Qt::white);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->show();

ui->label\_Info->setPalette(palette);

ui->label\_Info->setText("Surname area is empty. Please write your surname.");

*return*;

}

*else* *if*(ui->lineEdit\_password->text().isEmpty())

{

palette.setColor(ui->label\_Info->backgroundRole(), Qt::white);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->setPalette(palette);

ui->label\_Info->show();

ui->label\_Info->setText("Password area is empty. Please enter your password.");

*return*;

}

}

*//Перевіряє* *чи* *користувач* *є* *адміном*

void Loginpage::CheckLoginAdmin(QString surname, QString password)

{

*if*(!CheckAdmMode) *return*;

*if*(surname.toLower() == "admin" && password == "admin")

ShowAdminWindow();

*else* *if*(surname.toLower() != "admin")

SetWrongLoginMessage("login");

*else* *if*(password != "admin")

SetWrongLoginMessage("password");

}

*//Перевіряє* *чи* *користувач* *є* *вчителем*

void Loginpage::CheckLoginTeacher(QString surname, QString password)

{

*if*(!CheckTeachMode) *return*;

bool check = *false*;

*for*(int i = 0; i < teachersList.size(); ++i)

{

*if*(surname.toLower() == teachersList[i].surname.toLower() && password == teachersList[i].GetPassword())

{

check = *true*;

SetTeacherInfo(surname,password,i);

ShowTeacherWindow();

}

}

*if*(!check) SetWrongLoginMessage("surname or password");

}

*//Перевіряє* *чи* *користувач* *є* *учнем*

void Loginpage::CheckLoginStudent(QString surname, QString password)

{

*if*(!CheckStudMode) *return*;

bool check = *false*;

*for*(int i = 0; i < studentsList.size(); ++i)

{

*if*(surname.toLower() == studentsList[i].surname.toLower() && password == studentsList[i].GetPassword())

{

check = *true*;

SetStudentInfo(surname,password,i);

ShowStudentWindow();

}

}

*if*(!check) SetWrongLoginMessage("surname or password");

}

*//Виводить* *помилку* *входу* *із* *переданою* *стрічкою*

void Loginpage::SetWrongLoginMessage(QString whatsWrongString)

{

QPalette palette = ui->label\_Info->palette();

palette.setColor(ui->label\_Info->backgroundRole(), Qt::white);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->setPalette(palette);

ui->label\_Info->show();

ui->label\_Info->setText("Your " + whatsWrongString +" is wrong. Try again");

}

*//Відкриває* *вікно* *студента*

void Loginpage::ShowStudentWindow()

{

*this*->hide();

StudentPage page;

page.setWindowTitle("Student`s cabinet");

page.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\student.png"));

page.setModal(*true*);

page.exec();

}

*//Відкриває* *вікно* *викладача*

void Loginpage::ShowTeacherWindow()

{

*this*->hide();

TeacherPage page;

page.setWindowTitle("Teacher`s cabinet");

page.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\teacher.png"));

page.setModal(*true*);

page.exec();

}

*//Відкриває* *вікно* *адміна*

void Loginpage::ShowAdminWindow()

{

*this*->hide();

MainWindow window;

window.show();

}

*//Записує* *всю* *інформацію* *про* *користувача-викладача*

void Loginpage::SetTeacherInfo(QString surname, QString password, int index)

{

teacher.surname = surname;

teacher.SetPassword(password);

teacher.SetPosition(teachersList[index].GetPosition());

teacher.SetExperience(teachersList[index].GetExperience());

teacher.DeleteDisciplines();

*for*(int j = 0; j < teachersList[index].GetDisciplines().size(); ++j)

teacher.SetDiscipline(teachersList[index].GetDisciplines()[j]);

teacher.SetPersonalId(teachersList[index].GetPersonalId());

}

*//Записує* *всю* *інформацію* *про* *користувача-студента*

void Loginpage::SetStudentInfo(QString surname, QString password, int index)

{

QVector<QString> studTmpVect;

student.surname = surname;

student.SetPassword(password);

student.SetGroup(studentsList[index].GetGroup());

student.SetYear(studentsList[index].GetYear());

student.GetLessons().clear();

*for*(int j = 0; j < studentsList[index].GetLessons().size(); ++j)

studTmpVect.push\_back(studentsList[index].GetLessons()[j]);;

student.SetLessons(studTmpVect);

studTmpVect.clear();

student.SetPersonalId(studentsList[index].GetPersonalId());

*for*(int j = 0; j < studentsList[index].GetSizeOfIdOfPersons(); ++j)

studTmpVect.push\_back(studentsList[index].GetIdOfPersonsByIndex(j));

student.SetIdOfPersons(studTmpVect);

}

*//Відкиває* *вікно* *реєстраціїї*

void Loginpage::on\_commandLinkButton\_clicked()

{

hide();

Registrationpage window;

window.setWindowTitle("Registration");

window.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\registration.png"));

window.setModal(*true*);

window.exec();

}

*//Відкиває* *головне* *вікно* *програми*

void Loginpage::on\_commandLinkButton\_BackToTitle\_clicked()

{

hide();

MainWindow window;

CheckAdmMode = *false*;

window.show();

}

**~~~~~~~mainwindow.h~~~~~**

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <string>

#include <QDebug>

#include <QAction>

#include <QMenu>

#include <QComboBox>

#include <QTableWidget>

*using* *namespace* std;

QT\_BEGIN\_NAMESPACE

*namespace* **Ui** { *class* **MainWindow**; }

QT\_END\_NAMESPACE

*//Клас* *який* *відповідає* *за* *вигляд* *титульної* *сторінки* *і* *вікна* *адміністратора*

*class* **MainWindow** : *public* QMainWindow

{

Q\_OBJECT

*public*:

**MainWindow**(QWidget \*parent = *nullptr*);

~***MainWindow***();

*public* slots:

void **tableItemClicked**(int row, int column);

void **HideNewDisciplineWindowElements**();

void **HideElementsForTitleWindow**();

void **CheckReadFromTableInfo**();

void **CheckReadFromTableLineInfo**(int index);

void **SetNewTeacherInfo**(int index);

void **ShowDisciplines**(QComboBox \*disciplinesBox);

void **ShowNewDisciplineWindowElements**();

void **ShowColumnsHeaderName**();

*private* slots:

void **on\_pushButton\_Admin\_clicked**();

void **on\_pushButton\_Registration\_clicked**();

void **on\_pushButton\_Teacher\_clicked**();

void **on\_pushButton\_Student\_clicked**();

void **on\_pushButton\_OkSize\_clicked**();

void **on\_actionOpen\_triggered**();

void **on\_pushButton\_Set\_clicked**();

void **on\_pushButton\_AddDiscipline\_clicked**();

void **on\_actionAddTeacher\_triggered**();

void **on\_actionDeleteTeacher\_triggered**();

void **on\_actionCreateDiscipline\_triggered**();

void **on\_pushButton\_SetDiscipline\_clicked**();

void **on\_actionBackToTitle\_triggered**();

void **on\_commandLinkButton\_Close\_clicked**();

void **on\_pushButton\_SetNewTeacherInfo\_clicked**();

*private*:

Ui::MainWindow \*ui;

};

#endif *//* *MAINWINDOW\_H*

**~~~~~~~~mainwindow.cpp~~~~~~~~~**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "loginpage.h"

#include "registrationpage.h"

#include "Teacher.h"

#include "student.h"

#include "discipline.h"

#include "disciplinemanager.h"

#include "file.h"

bool CheckStudMode,CheckTeachMode,CheckAdmMode;

bool oneTimeRunningMode = *true*;

QList<Discipline> listOfDisciplines;

QList<Teacher> teachersList;

QList<Student> studentsList;

MainWindow::**MainWindow**(QWidget \*parent)

: QMainWindow(parent)

, ui(*new* Ui::MainWindow)

{

ui->setupUi(*this*);

*if*(oneTimeRunningMode)

{

File \*fileTmp = *new* File;

fileTmp->ReadAndCheckFilesInfo(*this*);

DisciplineManager \*disciplineManagerTmp = *new* DisciplineManager;

disciplineManagerTmp->DeleteAllInfoAboutOutdatedSubjects();

fileTmp->WriteToFileDisciplines("data\\Disciplines.txt",*this*);

ShowDisciplines(ui->comboBox\_Disciplines);

HideElementsForTitleWindow();

connect(ui->tableWidget\_Info, SIGNAL(cellDoubleClicked(int,int)), *this*, SLOT(tableItemClicked(int,int)));

oneTimeRunningMode = *false*;

*delete* fileTmp, disciplineManagerTmp;

}

}

MainWindow::~***MainWindow***()

{

File\* fileTmp = *new* File;

fileTmp->WriteToFileTeachers("data\\TeachersList.txt",*this*);

fileTmp->WriteToFileStudents("data\\StudentsList.txt",*this*);

*delete* fileTmp;

*delete* ui;

}

*//Виводить* *список* *дисциплін*

void MainWindow::**ShowDisciplines**(QComboBox \*disciplinesBox)

{

*for*(*auto* it = listOfDisciplines.begin(); it != listOfDisciplines.end(); ++it)

{

disciplinesBox->addItem(it->GetName().replace(" ","\_"));

it->SetName(it->GetName().replace("\_"," "));

}

}

*//Ховає* *елементи* *для* *показу* *титульного* *вікна*

void MainWindow::**HideElementsForTitleWindow**()

{

ui->menuBar->hide();

ui->toolBar->hide();

ui->tableWidget\_Info->hide();

ui->pushButton\_OkSize->hide();

ui->pushButton\_Set->hide();

ui->label\_TeacherInfo->hide();

ui->spinBox\_Size->hide();

ui->comboBox\_Disciplines->hide();

ui->label\_Disciplines->hide();

ui->pushButton\_AddDiscipline->hide();

HideNewDisciplineWindowElements();

}

*//Зчитує* *комірку,де* *виводяться* *дисципліни* *у* *таблиці,* *на* *яку* *користувач* *натиснув,*

*//Перевіряє* *чи* *даної* *дисципліни* *немає* *у* *вчителя,*

*//і* *також* *перевіряє* *чи* *нема* *вже* *у* *викладача* *дисциплін* *із* *даного* *курсу*

void MainWindow::**tableItemClicked**(int row, int column)

{

DisciplineManager \*tmp = *new* DisciplineManager;

*if*(column >= DISCIPLINE\_AREA)

{

QString str = ui->comboBox\_Disciplines->currentText();

QTableWidgetItem \*item = *new* QTableWidgetItem();

item->setFlags(item->flags() ^ Qt::*ItemIsEditable*);

*for*(int i = DISCIPLINE\_AREA; i < ui->tableWidget\_Info->columnCount(); ++i)

{

item = ui->tableWidget\_Info->item(row,i);

*if* (!item || item->text().isEmpty())

*break*;

*if*(tmp->FindCourseOfSubject(str) == tmp->FindCourseOfSubject(item->text()))

*return*;

}

ui->tableWidget\_Info->setItem(row,column,*new* QTableWidgetItem(str));

ui->tableWidget\_Info->clearSelection();

}

}

*//Приховує* *елементи* *розділу* *створення* *нової* *дисципліни*

void MainWindow::**HideNewDisciplineWindowElements**()

{

ui->commandLinkButton\_Close->hide();

ui->label\_DisCourse->hide();

ui->label\_DisName->hide();

ui->lineEdit\_DisciplineName->hide();

ui->comboBox\_DisCourse->hide();

ui->pushButton\_SetDiscipline->hide();

ui->label\_EndDate->hide();

ui->label\_StartDate->hide();

ui->checkBox\_Mon->hide();

ui->checkBox\_Tue->hide();

ui->checkBox\_Wed->hide();

ui->checkBox\_Thu->hide();

ui->checkBox\_Fri->hide();

ui->dateEdit\_Start->hide();

ui->dateEdit\_End->hide();

ui->label\_Days->hide();

ui->line->hide();

ui->pushButton\_SetNewTeacherInfo->hide();

}

*//Відкриває* *вікно* *входу* *в* *систему* *як* *адмін*

void MainWindow::**on\_pushButton\_Admin\_clicked**()

{

CheckAdmMode = *true*;

hide();

Loginpage window;

window.setModal(*true*);

window.*exec*();

show();

*if*(CheckAdmMode)

{

ui->tableWidget\_Info->show();

ui->menuBar->show();

ui->toolBar->show();

ui->pushButton\_OkSize->show();

ui->label\_TeacherInfo->show();

ui->spinBox\_Size->show();

ui->label\_Disciplines->show();

ui->comboBox\_Disciplines->show();

ui->pushButton\_Admin->hide();

ui->pushButton\_Student->hide();

ui->pushButton\_Teacher->hide();

ui->pushButton\_Registration->hide();

ui->label\_Title->hide();

ui->label\_Titlelable->hide();

}

}

*//Відкриває* *вікно* *реєстрації*

void MainWindow::**on\_pushButton\_Registration\_clicked**()

{

hide();

Registrationpage window;

window.setWindowTitle("Registration");

window.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\registration.png"));

window.setModal(*true*);

window.*exec*();

show();

}

*//Відкриває* *вікно* *входу* *користувача* *як* *викладача*

void MainWindow::**on\_pushButton\_Teacher\_clicked**()

{

CheckTeachMode = *true*;

hide();

Loginpage window;

window.setWindowTitle("Login");

window.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\login.png"));

window.setModal(*true*);

window.*exec*();

show();

}

*//Відкриває* *вікно* *входу* *користувача* *як* *студента*

void MainWindow::**on\_pushButton\_Student\_clicked**()

{

CheckStudMode = *true*;

hide();

Loginpage window;

window.setWindowTitle("Login");

window.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\login.png"));

window.setModal(*true*);

window.*exec*();

show();

}

*//Зчитує* *і* *виводить* *ту* *кількість* *рядків* *для* *вводу* *інформації* *про* *вчителів*

void MainWindow::**on\_pushButton\_OkSize\_clicked**()

{

int size = ui->spinBox\_Size->text().toInt();

ui->tableWidget\_Info->setRowCount(size);

ShowColumnsHeaderName();

ui->pushButton\_AddDiscipline->show();

ui->pushButton\_SetNewTeacherInfo->show();

ui->pushButton\_Set->show();

}

*//Перевіряє* *і* *зберігає* *введену* *інформацію* *про* *вчителів* *з* *таблиці*

void MainWindow::**on\_pushButton\_Set\_clicked**()

{

teachersList.clear();

*try*

{

CheckReadFromTableInfo();

} *catch* (exception &ex) {

QMessageBox::critical(*this*,"Wrong information format",ex.*what*());

*return*;

}

*for*(int i = 0; i < ui->tableWidget\_Info->rowCount(); ++i)

{

Teacher tmp;

tmp.surname = ui->tableWidget\_Info->item(i,0)->text();

tmp.SetExperience(ui->tableWidget\_Info->item(i,1)->text().toInt());

QPair<int,QString> tmpPos;

tmpPos.second = ui->tableWidget\_Info->item(i,2)->text();

tmp.SetPosition(tmpPos);

tmp.SetPriorityToTeacherPosition();

tmp.SetPersonalId(ui->tableWidget\_Info->item(i,3)->text());

tmp.SetPassword(ui->tableWidget\_Info->item(i,4)->text());

*for*(int j = DISCIPLINE\_AREA, counter = 0; j < ui->tableWidget\_Info->columnCount(); ++j)

{

QTableWidgetItem \*itm = ui->tableWidget\_Info->item(i,j);

*if* (!itm || itm->text().isEmpty()) *break*;

*else*

{

Discipline tmpDiscip;

tmpDiscip.SetName(ui->tableWidget\_Info->item(i,j)->text());

tmp.SetDiscipline(tmpDiscip);

++counter;

}

}

teachersList.push\_back(tmp);

}

QMessageBox::information(*this*,"Set Information", "Successfully saved information");

}

*//Виводить* *заголовки* *колонок* *таблиці*

void MainWindow::**ShowColumnsHeaderName**()

{

ui->tableWidget\_Info->setColumnCount(6);

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("EXPERIENCE(years)"));

ui->tableWidget\_Info->setHorizontalHeaderItem(2, *new* QTableWidgetItem("POSITION"));

ui->tableWidget\_Info->setHorizontalHeaderItem(3, *new* QTableWidgetItem("ID"));

ui->tableWidget\_Info->setHorizontalHeaderItem(4, *new* QTableWidgetItem("PASSWORD"));

ui->tableWidget\_Info->setHorizontalHeaderItem(5, *new* QTableWidgetItem("DISCIPLINE"));

}

*//Викликає* *функцію* *для* *вибору* *файла* *із* *викладачами* *та* *зчитує* *інформацію* *з* *нього*

void MainWindow::**on\_actionOpen\_triggered**()

{

ui->pushButton\_Set->show();

File \*fileTmp = *new* File;

fileTmp->OpenFileWithChoise(*this*);

QFile file(fileTmp->path);

QString line,tmp;

QRegExp symbols("(\\ |\\,|\\.|\\:|\\t+|\\s+)");

*try*

{

fileTmp->CheckIfOpenFile(*file*, fileTmp->path);

} *catch* (exception &ex)

{

QMessageBox::critical(*this*,"Pr school:File error",ex.*what*());

}

QTextStream input(&file);

int counter = 0,max = 0;

*while*(!input.atEnd())

{

tmp = input.readLine();

QStringList lineList = tmp.split(symbols);

*if*(max < lineList.length())

max = lineList.length();

++counter;

}

file.*close*();

file.*open*(QFile::*ReadOnly* | QFile::*Text*);

ui->tableWidget\_Info->setRowCount(counter);

ui->tableWidget\_Info->setColumnCount(max);

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("EXPERIENCE(years)"));

ui->tableWidget\_Info->setHorizontalHeaderItem(2, *new* QTableWidgetItem("POSITION"));

ui->tableWidget\_Info->setHorizontalHeaderItem(3, *new* QTableWidgetItem("ID"));

ui->tableWidget\_Info->setHorizontalHeaderItem(4, *new* QTableWidgetItem("PASSWORD"));

*for*(int i = DISCIPLINE\_AREA; i < ui->tableWidget\_Info->columnCount(); ++i)

ui->tableWidget\_Info->setHorizontalHeaderItem(i, *new* QTableWidgetItem("DISCIPLINE"));

ui->pushButton\_AddDiscipline->show();

ui->pushButton\_Set->show();

int j = 0;

*for* (int i = 0; i < counter; ++i)

{

line = input.readLine();

QStringList query = line.split(symbols);

*for*(int i = 0; i < query.size(); ++i)

ui->tableWidget\_Info->setItem(j,i,*new* QTableWidgetItem(query.at(i)));

++j;

}

file.*close*();

*delete* fileTmp;

}

*//Добавляє* *поле* *для* *вводу* *нової* *дисципліни*

void MainWindow::**on\_pushButton\_AddDiscipline\_clicked**()

{

ui->tableWidget\_Info->setColumnCount(ui->tableWidget\_Info->columnCount()+1);

ui->tableWidget\_Info->setHorizontalHeaderItem(ui->tableWidget\_Info->columnCount()-1, *new* QTableWidgetItem("DISCIPLINE"));

}

*//Добавляє* *поле* *для* *вводу* *нового* *викладача*

void MainWindow::**on\_actionAddTeacher\_triggered**()

{

ui->tableWidget\_Info->setRowCount(ui->tableWidget\_Info->rowCount()+1);

ui->pushButton\_AddDiscipline->show();

ui->pushButton\_SetNewTeacherInfo->show();

}

*//Видаляє* *останнє* *поле* *таблиці*

void MainWindow::**on\_actionDeleteTeacher\_triggered**()

{

ui->tableWidget\_Info->setRowCount(ui->tableWidget\_Info->rowCount()-1);

ui->pushButton\_AddDiscipline->show();

}

*//Викликає* *функцію* *для* *показу* *розділу* *із* *створенням* *нової* *дисципліни*

void MainWindow::**on\_actionCreateDiscipline\_triggered**()

{

QRegExp rx("[a-zA-Z]+");

QValidator \*validator = *new* QRegExpValidator(rx, *this*);

ui->lineEdit\_DisciplineName->setValidator(validator);

ShowNewDisciplineWindowElements();

}

*//Виводить* *розділ* *із* *створенням* *нової* *дисципліни*

void MainWindow::**ShowNewDisciplineWindowElements**()

{

ui->label\_DisCourse->show();

ui->label\_DisName->show();

ui->lineEdit\_DisciplineName->show();

ui->comboBox\_DisCourse->show();

ui->pushButton\_SetDiscipline->show();

ui->label\_EndDate->show();

ui->label\_StartDate->show();

ui->checkBox\_Mon->show();

ui->checkBox\_Tue->show();

ui->checkBox\_Wed->show();

ui->checkBox\_Thu->show();

ui->checkBox\_Fri->show();

ui->dateEdit\_Start->show();

ui->dateEdit\_End->show();

ui->label\_Days->show();

ui->commandLinkButton\_Close->show();

ui->line->show();

}

*//Перевіряє* *та* *зберігає* *інформацію* *про* *нову* *дисципліну*

void MainWindow::**on\_pushButton\_SetDiscipline\_clicked**()

{

ui->comboBox\_Disciplines->addItem(ui->lineEdit\_DisciplineName->text());

int course = ui->comboBox\_DisCourse->currentText().toInt();

Discipline tmpDiscipline;

tmpDiscipline.SetCourse(course);

tmpDiscipline.SetName(ui->lineEdit\_DisciplineName->text());

tmpDiscipline.SetStartDate(ui->dateEdit\_Start->text());

tmpDiscipline.SetEndDate(ui->dateEdit\_End->text());

QVector<QString> daysForDiscipline;

*if*(ui->checkBox\_Mon->isChecked())

daysForDiscipline.push\_back("Mon");

*if*(ui->checkBox\_Tue->isChecked())

daysForDiscipline.push\_back("Tue");

*if*(ui->checkBox\_Wed->isChecked())

daysForDiscipline.push\_back("Wed");

*if*(ui->checkBox\_Thu->isChecked())

daysForDiscipline.push\_back("Thu");

*if*(ui->checkBox\_Fri->isChecked())

daysForDiscipline.push\_back("Fri");

tmpDiscipline.SetDaysInWeek(daysForDiscipline);

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(listOfDisciplines[i].GetName() == tmpDiscipline.GetName())

{

QMessageBox::critical(*this*,"Add new discipline","This discipline already exists");

*return*;

}

listOfDisciplines.push\_back(tmpDiscipline);

QDate start = QDate::fromString(tmpDiscipline.GetStartDate(),"dd/MM/yyyy");

QDate end = QDate::fromString(tmpDiscipline.GetEndDate(),"dd/MM/yyyy");

*if*(tmpDiscipline.GetName() == "")

{

QMessageBox::critical(*this*,"DISCIPLINE","Enter name of discipline");

*return*;

}

*else* *if*(end <= start)

{

QMessageBox::critical(*this*,"DISCIPLINE","End date must be more than start date");

*return*;

}

*else* *if*(!daysForDiscipline.size())

{

QMessageBox::critical(*this*,"DISCIPLINE","Choose days for discipline");

*return*;

}

*else*

QMessageBox::information(*this*, "Add new discipline", "Successful added new discipline to Discipline list");

QString filePathDisciplines = "data\\Disciplines.txt";

File \*tmpFile = *new* File;

tmpFile->WriteToFileNewDiscipline(filePathDisciplines,*this*);

HideNewDisciplineWindowElements();

*delete* tmpFile;

}

*//Ховає* *елементи* *вікна* *для* *виводу* *титульної* *сторінки* *програми*

void MainWindow::**on\_actionBackToTitle\_triggered**()

{

ui->pushButton\_Admin->show();

ui->pushButton\_Student->show();

ui->pushButton\_Teacher->show();

ui->pushButton\_Registration->show();

ui->label\_Title->show();

ui->label\_Titlelable->show();

HideNewDisciplineWindowElements();

ui->tableWidget\_Info->hide();

ui->menuBar->hide();

ui->toolBar->hide();

ui->pushButton\_OkSize->hide();

ui->label\_TeacherInfo->hide();

ui->spinBox\_Size->hide();

ui->label\_Disciplines->hide();

ui->comboBox\_Disciplines->hide();

ui->pushButton\_SetNewTeacherInfo->hide();

ui->pushButton\_Set->hide();

ui->pushButton\_AddDiscipline->hide();

}

*//Ховає* *елементи* *розділу* *для* *добавлення* *нової* *дисципліни*

void MainWindow::**on\_commandLinkButton\_Close\_clicked**()

{

HideNewDisciplineWindowElements();

}

*//Шукає* *нових* *викладачів* *із* *таблиці* *та* *викликає* *функцію* *що* *зберігає* *дані* *про* *них*

void MainWindow::**on\_pushButton\_SetNewTeacherInfo\_clicked**()

{

QVector<int> indexesOfNewTeachers;

*for*(int i = 0; i < ui->tableWidget\_Info->rowCount(); ++i)

{

bool checkIfRegisteredString = *false*;

*for*(int j = 0; j < teachersList.size(); ++j)

*if*(ui->tableWidget\_Info->item(i,0)->text() == teachersList[j].surname)

checkIfRegisteredString = *true*;

*if*(!checkIfRegisteredString)

indexesOfNewTeachers.push\_back(i);

}

*if*(!indexesOfNewTeachers.size())

{

QMessageBox::warning(*this*,"Add new teachers","There aren`t new teachers in table");

*return*;

}

int oldSizeTeacherList,newSizeTeacherList;

oldSizeTeacherList = teachersList.size();

*for*(int i = 0; i < indexesOfNewTeachers.size(); ++i)

SetNewTeacherInfo(indexesOfNewTeachers.at(i));

newSizeTeacherList = teachersList.size();

*if*(newSizeTeacherList - oldSizeTeacherList > 0)

{

QMessageBox::information(*this*,"Add new teachers","Successfully added " +

QString::number(indexesOfNewTeachers.size()) + " teacher(s)");

}

}

*//Перевіряє* *та* *зберігає* *інформацію* *про* *нового* *викладача*

void MainWindow::**SetNewTeacherInfo**(int index)

{

*try* {

CheckReadFromTableLineInfo(index);

} *catch* (exception &ex) {

QMessageBox::warning(*this*,"Add new teacher",ex.*what*());

*return*;

}

Teacher tmp;

tmp.surname = ui->tableWidget\_Info->item(index,0)->text();

tmp.SetExperience(ui->tableWidget\_Info->item(index,1)->text().toInt());

QPair<int,QString> tmpPos;

tmpPos.second = ui->tableWidget\_Info->item(index,2)->text();

tmp.SetPosition(tmpPos);

tmp.SetPriorityToTeacherPosition();

tmp.SetPersonalId(ui->tableWidget\_Info->item(index,3)->text());

tmp.SetPassword(ui->tableWidget\_Info->item(index,4)->text());

*for*(int i = DISCIPLINE\_AREA; i < ui->tableWidget\_Info->columnCount(); ++i)

{

QTableWidgetItem \*itm = ui->tableWidget\_Info->item(index,i);

*if* (!itm || itm->text().isEmpty()) *break*;

*else*

{

QString discipName;

discipName = ui->tableWidget\_Info->item(index,i)->text();

*for*(int j = 0; j < listOfDisciplines.size(); ++j)

{

*if*(discipName == listOfDisciplines[j].GetName().replace(" ","\_"))

{

tmp.SetDiscipline(listOfDisciplines[j]);

teachersList.push\_back(tmp);

*return*;

}

}

}

}

}

*//Перевіряє* *правильність* *введеної* *інформації* *у* *таблиці*

void MainWindow::**CheckReadFromTableInfo**()

{

*for*(int i = 0; i < ui->tableWidget\_Info->rowCount(); ++i)

CheckReadFromTableLineInfo(i);

}

*//Перевіряє* *правильність* *введеної* *інформації* *у* *рядку* *таблиці*

void MainWindow::**CheckReadFromTableLineInfo**(int index)

{

QRegExp surnameCheck("(\\d\*|\\W\*)");*//* *a* *digit* *(\d),* *zero* *or* *more* *times* *(\*),* *(\w\*)* *word* *character* *more* *than* *1*

QRegExp idAndExperienceCheck("\\D\*");

Teacher tmp;

tmp.surname = ui->tableWidget\_Info->item(index,0)->text();

*if* (surnameCheck.exactMatch(tmp.surname) || tmp.surname.isEmpty())

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " surname wrong");

QString experience = ui->tableWidget\_Info->item(index,1)->text();

*if* (idAndExperienceCheck.exactMatch(experience) || experience.isEmpty())

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " experience wrong");

QPair<int,QString> tmpPos;

tmpPos.second = ui->tableWidget\_Info->item(index,2)->text();

tmp.SetPosition(tmpPos);

tmp.SetPriorityToTeacherPosition();

*if*(tmp.GetPosition().first < 0)

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " position wrong");

QString teacherID = ui->tableWidget\_Info->item(index,3)->text();

*for*(int i = 0; i < teachersList.size(); ++i)

*if*(teacherID == teachersList[i].GetPersonalId())

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " personal id is not unique");

tmp.SetPersonalId(teacherID);

*if* (idAndExperienceCheck.exactMatch(tmp.GetPersonalId()) || tmp.GetPersonalId() == "")

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " id wrong");

tmp.SetPassword(ui->tableWidget\_Info->item(index,4)->text());

*if*(tmp.GetPassword() == "")

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " password is empty");

*for*(int j = DISCIPLINE\_AREA; j < ui->tableWidget\_Info->columnCount(); ++j)

{

QTableWidgetItem \*itm = ui->tableWidget\_Info->item(index,j);

*if* (!itm || itm->text().isEmpty()) *break*;

QString checkString = itm->text();

*for*(int z = 0; z < checkString.size(); ++z)

{

*if*(!checkString[z].isLetter() && checkString[z] != '\_' && checkString[z] != '-')

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " discipline format wrong");

}

bool check = *false*;

*for*(int k = 0; k < listOfDisciplines.size(); ++k)

{

QString name = listOfDisciplines[k].GetName();

QString formatName = name.replace(" ","\_");

*if*(formatName == ui->tableWidget\_Info->item(index,j)->text())

check = *true*;

}

*if*(!check)

*throw* runtime\_error("In line " + QString::number(index+1).toStdString() + " discipline wrong");

}

}

**~~~~~~~~~registrationpage.h~~~~~~~~**

#ifndef REGISTRATIONPAGE\_H

#define REGISTRATIONPAGE\_H

#include <QDialog>

#include "student.h"

#include "discipline.h"

#include "loginpage.h"

#include <random>

#include <ctime>

*//Клас,* *що* *відповідає* *за* *вікно* *реєстрації*

*namespace* Ui {

*class* Registrationpage;

}

*class* Registrationpage : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* Registrationpage(QWidget \*parent = *nullptr*);

void CheckIfStudentAlreadyInList(Student regStudent);

void CheckInput();

~Registrationpage();

*private* *slots*:

void on\_pushButton\_Reg\_clicked();

void on\_commandLinkButton\_BackToLogin\_clicked();

*private*:

Ui::Registrationpage \*ui;

};

#endif *//* *REGISTRATIONPAGE\_H*

**~~~~~~registrationpage.cpp~~~~~~~~**

#include "registrationpage.h"

#include "ui\_registrationpage.h"

Registrationpage::Registrationpage(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::Registrationpage)

{

ui->setupUi(*this*);

QRegExp rx("[a-zA-Z]+");

QValidator \*validator = *new* QRegExpValidator(rx, *this*);

ui->lineEdit\_Surname->setValidator(validator);

ui->lineEdit\_Surname->setPlaceholderText("Your Surname");

ui->lineEdit\_GroupName->setPlaceholderText("Your Group Name");

ui->lineEdit\_Password->setPlaceholderText("Password");

ui->lineEdit\_Password2->setPlaceholderText("Confirm your Password");

ui->lineEdit\_GroupName->setValidator(validator);

}

Registrationpage::~Registrationpage()

{

*delete* ui;

}

*//Викликає* *ф-ю,* *що* *перевіряє* *дані.*

*//Зберігає* *дані* *введені* *у* *поля* *реєстрації*

void Registrationpage::on\_pushButton\_Reg\_clicked()

{

CheckInput();

Student person;

person.surname = ui->lineEdit\_Surname->text();

QStringList parts = person.surname.toLower().split(' ', QString::SkipEmptyParts);

*for* (int i = 0; i < parts.size(); ++i)

parts[i].replace(0, 1, parts[i][0].toUpper());

person.surname = parts.join(" ");

person.SetYear(ui->comboBox\_Course->currentText().toInt());

person.SetGroup(ui->lineEdit\_GroupName->text().toUpper() + "-" +

QString::number(person.GetYear()) + ui->comboBox\_GroupNum->currentText());

person.SetDisciplinesToStudent(person,person.GetYear());

mt19937 engine;

engine.seed(time(*nullptr*));

person.SetPersonalId(QString::number(engine() %1000000 + 1));

QPalette palette = ui->label\_Info->palette();

QString password;

*if*(ui->lineEdit\_Password->text() == ui->lineEdit\_Password2->text())

{

password = ui->lineEdit\_Password2->text();

person.SetPassword(password);

}

*else*

{

palette.setColor(ui->label\_Info->backgroundRole(), Qt::white);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->setPalette(palette);

ui->label\_Info->setText("Your password inputs are not equel. Try again.");

*return*;

}

*try*

{

CheckIfStudentAlreadyInList(person);

} *catch* (exception &ex)

{

QMessageBox::warning(*this*,"Registration",ex.what());

*return*;

}

studentsList.push\_back(person);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::darkGreen);

ui->label\_Info->setPalette(palette);

ui->label\_Info->setText("You are successfully signed up!");

}

*//Перевіряє* *введені* *у* *вікні* *дані*

void Registrationpage::CheckInput()

{

QPalette palette = ui->label\_Info->palette();

*if*(ui->lineEdit\_Surname->text().isEmpty() ||

ui->lineEdit\_GroupName->text().isEmpty() ||

ui->lineEdit\_Password->text().isEmpty() ||

ui->lineEdit\_Password2->text().isEmpty())

{

palette.setColor(ui->label\_Info->backgroundRole(), Qt::white);

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->setPalette(palette);

ui->label\_Info->setText("Some of registration lines are empty. Fill empty lines.");

*return*;

}

QString str = ui->lineEdit\_Password->text();

*if*(str.size() < 5)

{

palette.setColor(ui->label\_Info->foregroundRole(), Qt::red);

ui->label\_Info->setPalette(palette);

ui->label\_Info->setText("Your password should be at least 5 characters. Try again.");

*return*;

}

}

*//Перевіряє* *чи* *зареєстрований* *студент* *є* *унікальним*

void Registrationpage::CheckIfStudentAlreadyInList(Student regStudent)

{

*for*(int i = 0; i < studentsList.size(); ++i)

*if*(regStudent.surname == studentsList.at(i).surname && regStudent.GetGroup() == studentsList[i].GetGroup())

*throw* runtime\_error("This student is already registered");

}

*//Відкриває* *вікно* *входу* *в* *систему*

void Registrationpage::on\_commandLinkButton\_BackToLogin\_clicked()

{

hide();

Loginpage page;

CheckStudMode = *true*;

page.setModal(*true*);

page.exec();

}

**~~~~~~~~student.h~~~~~~~~~~~~~**

#ifndef STUDENT\_H

#define STUDENT\_H

#include <QFile>

#include <QFileDialog>

#include <QComboBox>

#include <QTableWidget>

#include <QDate>

#include <QCalendarWidget>

#include <QLocale>

#include <QString>

#include <QVector>

#include "Teacher.h"

#include "discipline.h"

*//Клас* *Студент,* *який* *містить* *рік* *навчання,* *групу* *та* *предмети* *відповідно* *до* *курсу*

*class* Student:*public* Human

{

*public*:

Student();

~Student(){};

void SetYear(int year);

void SetGroup(QString group);

void SetLessons(QVector<QString> lessons);

int GetYear();

QString GetGroup();

QVector<QString> GetLessons();

void ChooseTeacher(QString myTeacherID);

int FindTheMostPopularTeacher(QList<Teacher> teachersList, QComboBox \*subject, Teacher &resTeacher);

void SortTeacherListBySurname(QList<Teacher> &teachersList);

void SortTeacherListByPosition(QList<Teacher> &teachersList);

void FindTeachersWhoHaveOneSubject(QList<Teacher> teachersList,QList<Teacher> &resList);

void SetDisciplinesToStudent(Student &student,int year);

bool *operator*<(Student &another)

{

*return* *this*->surname < another.surname;

}

bool *operator*>(Student &another)

{

*return* *this*->group > another.group;

}

*private*:

int year;

QString group;

QVector<QString> lessons;

};

*//Глобальний* *список* *студентів*

*extern* QList<Student> studentsList;

*//Глобальна* *змінна* *студента-користувача*

*extern* Student student;

#endif *//* *STUDENT\_H*

**~~~~~~~~studentMethods.cpp~~~~~~**

#include "student.h"

#define MAX\_NAME 32

Student::Student(){}

void Student::SetYear(int year)

{

*this*->year = year;

}

void Student::SetGroup(QString group)

{

*this*->group = group;

}

void Student::SetLessons(QVector<QString> lessons)

{

*this*->lessons = lessons;

}

int Student::GetYear()

{

*return* year;

}

QString Student::GetGroup()

{

*return* group;

}

QVector<QString> Student::GetLessons()

{

*return* lessons;

}

*//Сортує* *всіх* *вчителів* *за* *посадою* *(від* *найважливішої* *до* *найменш* *важливої)*

void Student::SortTeacherListByPosition(QList<Teacher> &teachersList)

{

sort(teachersList.begin(),teachersList.end(),[](Teacher &firstTeacherPosition, Teacher &secondTeacherPosition){*return* firstTeacherPosition > secondTeacherPosition;});

}

*//Сортує* *всіх* *вчителів* *за* *прізвищем* *(в* *алфавітному* *порядку)*

void Student::SortTeacherListBySurname(QList<Teacher> &teachersList)

{

sort(teachersList.begin(),teachersList.end(),[] (Teacher &firstTeacherSurname, Teacher &secondTeacherSurname){*return* firstTeacherSurname < secondTeacherSurname;});

}

*//Знаходить* *серед* *всіх* *вчителів,* *тих* *хто* *має* *тільки* *одну* *дисципліну*

void Student::FindTeachersWhoHaveOneSubject(QList<Teacher> teachersList,QList<Teacher> &resList)

{

*for*(int i = 0; i < teachersList.size(); ++i)

*if*(teachersList[i].GetDisciplines().size() == 1)

{

Teacher tmp;

tmp.surname = teachersList[i].surname;

tmp.SetPosition(teachersList[i].GetPosition());

tmp.SetDiscipline(teachersList[i].GetDisciplines()[0]);

resList.push\_back(tmp);

}

}

*//Знаходить* *серед* *всіх* *вчителів* *вибраної* *дисципліни* *найпопулярнішого(найбільше* *записаних* *студентів)*

int Student::FindTheMostPopularTeacher(QList<Teacher> teachersList, QComboBox \*subject, Teacher &resTeacher)

{

QVector<Teacher> thisDiscipTeachers;

int counter = 0;

*if*(subject->currentText() == "")

*return* -1;

*for*(int i = 0; i < teachersList.size(); ++i)

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

*if*(teachersList[i].GetDisciplines()[j].GetName() == subject->currentText())

thisDiscipTeachers.push\_back(teachersList[i]);

int max = thisDiscipTeachers[0].GetSizeOfIdOfPersons();

resTeacher.surname = thisDiscipTeachers[0].surname;

resTeacher.SetPosition(thisDiscipTeachers[0].GetPosition());

counter = thisDiscipTeachers[0].GetSizeOfIdOfPersons();

*for*(int i = 0; i < thisDiscipTeachers[0].GetDisciplines().size(); ++i)

resTeacher.SetDiscipline(thisDiscipTeachers[0].GetDisciplines()[i]);

*for*(int i = 1; i < thisDiscipTeachers.size(); ++i)

{

*if*(max < thisDiscipTeachers[i].GetSizeOfIdOfPersons())

{

max = thisDiscipTeachers[i].GetSizeOfIdOfPersons();

resTeacher.surname = thisDiscipTeachers[i].surname;

resTeacher.SetPosition(thisDiscipTeachers[i].GetPosition());

counter = thisDiscipTeachers[i].GetSizeOfIdOfPersons();

*for*(int i = 0; i < thisDiscipTeachers[i].GetDisciplines().size(); ++i)

resTeacher.SetDiscipline(thisDiscipTeachers[i].GetDisciplines()[i]);

}

}

*return* counter;

}

*//Записує* *дані* *студенту* *про* *обраного* *вчителя* *на* *задану* *дисципліну*

void Student::ChooseTeacher(QString myTeacherID)

{

student.SetIdOfPersonsByIndex(myTeacherID,student.GetSizeOfIdOfPersons());

}

*//Записує* *дані* *про* *дисципліни* *студенту* *відповідно* *до* *його* *курсу*

void Student::SetDisciplinesToStudent(Student &student,int year)

{

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

*if*(listOfDisciplines[i].GetCourse() == year)

student.lessons.push\_back(listOfDisciplines[i].GetName());

}

**~~~~~studentpage.h~~~~~~**

#ifndef STUDENTPAGE\_H

#define STUDENTPAGE\_H

#include <QDialog>

#include <QTableWidget>

#include "Teacher.h"

*//Клас* *що* *відповідає* *за* *вікно* *студента*

*namespace* **Ui** {

*class* **StudentPage**;

}

*class* **StudentPage** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **StudentPage**(QWidget \*parent = *nullptr*);

~***StudentPage***();

*public* slots:

void **tableItemClicked**(int row, int column);

void **PrintInfo**(QTableWidget \*tableTeacherInfo, QList<Teacher> list, int mode);

void **ShowTeachersListInfo**(QString subject, QTableWidget \*table);

void **ShowMyTeachersWithDisciplines**();

void **ShowMyTeachers**();

void **ShowMyDisciplines**();

*private* slots:

void **on\_pushButton\_SortSurname\_clicked**();

void **on\_pushButton\_SortPosition\_clicked**();

void **on\_pushButton\_Teacher1Disc\_clicked**();

void **on\_pushButton\_FindTheMostPopularTeacher\_clicked**();

void **on\_pushButton\_ShowTeachers\_clicked**();

void **on\_pushButton\_ShowMyDisTeachers\_clicked**();

void **on\_pushButton\_ChooseTeacher\_clicked**();

void **on\_pushButton\_FindFreeTeachersByDate\_clicked**();

void **on\_commandLinkButton\_BackLink\_clicked**();

void **on\_tableWidget\_Info\_customContextMenuRequested**(*const* QPoint &pos);

*private*:

Ui::StudentPage \*ui;

};

#endif *//* *STUDENTPAGE\_H*

**~~~~~~~~~studentpage.cpp~~~~~~~**

#include "studentpage.h"

#include "ui\_studentpage.h"

#include "mainwindow.h"

#include "loginpage.h"

#include "student.h"

#include "discipline.h"

#include "disciplinemanager.h"

#define PRINT\_DISCIPLINE\_MODE 2

#define PRINT\_COURSE 1

StudentPage::StudentPage(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::StudentPage)

{

ui->setupUi(*this*);

QStringList parts = student.surname.toLower().split(' ', QString::SkipEmptyParts);

*for* (int i = 0; i < parts.size(); ++i)

parts[i].replace(0, 1, parts[i][0].toUpper());

ui->label\_Surname->setText("Welcome " + parts.join(" "));

ui->tableWidget\_Disciplines->setColumnCount(2);

ui->tableWidget\_Disciplines->setRowCount(student.GetLessons().size());

ui->calendarWidget->showToday();

*for*(int i = 0; i < listOfDisciplines.size(); ++i)

ui->comboBox\_Disciplines->addItem(listOfDisciplines[i].GetName());

*if*(student.GetSizeOfIdOfPersons() == NUMS\_OF\_DISCIPLINES)

{

ShowMyTeachersWithDisciplines();

}

*else* *if*(student.GetSizeOfIdOfPersons())

{

connect(ui->tableWidget\_Info, *SIGNAL*(cellDoubleClicked(int,int)), *this*, *SLOT*(tableItemClicked(int,int)));

ShowMyDisciplines();

ShowMyTeachers();

}

*else*

{

connect(ui->tableWidget\_Info, *SIGNAL*(cellDoubleClicked(int,int)), *this*, *SLOT*(tableItemClicked(int,int)));

ShowMyDisciplines();

}

}

StudentPage::~StudentPage()

{

*delete* ui;

}

*//Виводить* *інформацію* *про* *дисципліни* *студента* *та* *відповідними* *його* *вчителями*

void StudentPage::ShowMyTeachersWithDisciplines()

{

QList<Teacher> myTeachers;

ui->pushButton\_ChooseTeacher->hide();

*for*(int i = 0; i < teachersList.size(); ++i)

*for*(int j = 0; j < student.GetSizeOfIdOfPersons(); ++j)

*if*(teachersList[i].GetPersonalId() == student.GetIdOfPersonsByIndex(j))

myTeachers.push\_back(teachersList[i]);

ui->tableWidget\_Disciplines->setHorizontalHeaderItem(0, *new* QTableWidgetItem("MY DISCIPLINES"));

ui->tableWidget\_Disciplines->setHorizontalHeaderItem(1, *new* QTableWidgetItem("SURNAMES"));

*for*(int k = 0; k < student.GetLessons().size(); ++k)

*for*(int i = 0; i < myTeachers.size(); ++i)

*for*(int j = 0; j < myTeachers[i].GetDisciplines().size(); ++j)

{

*if*(student.GetLessons()[k] == myTeachers[i].GetDisciplines()[j].GetName().replace("\_"," "))

{

ui->comboBox\_MyDisciplines->addItem(student.GetLessons()[k]);

ui->tableWidget\_Disciplines->setItem(k,0,*new* QTableWidgetItem(student.GetLessons()[k]));

ui->tableWidget\_Disciplines->setItem(k,1,*new* QTableWidgetItem(myTeachers[i].surname));

}

}

}

*//Виводить* *інформацію* *про* *дисципліни* *студента*

void StudentPage::ShowMyDisciplines()

{

ui->tableWidget\_Disciplines->setHorizontalHeaderItem(0, *new* QTableWidgetItem("MY DISCIPLINES"));

ui->tableWidget\_Disciplines->setHorizontalHeaderItem(1, *new* QTableWidgetItem("SURNAMES"));

*for*(int k = 0; k < student.GetLessons().size(); ++k)

{

ui->comboBox\_MyDisciplines->addItem(student.GetLessons()[k]);

ui->tableWidget\_Disciplines->setItem(k,0,*new* QTableWidgetItem(student.GetLessons()[k]));

}

}

*//Виводить* *інформацію* *про* *викладачів* *студента*

void StudentPage::ShowMyTeachers()

{

ui->pushButton\_ChooseTeacher->show();

QList<Teacher> myTeachers;

*for*(int i = 0; i < teachersList.size(); ++i)

*for*(int j = 0; j < student.GetSizeOfIdOfPersons(); ++j)

*if*(teachersList[i].GetPersonalId() == student.GetIdOfPersonsByIndex(j))

myTeachers.push\_back(teachersList[i]);

*for*(int i = 0; i < myTeachers.size(); ++i)

ui->tableWidget\_Disciplines->setItem(i,1,*new* QTableWidgetItem(myTeachers[i].surname));

}

*//Зчитує* *комірку* *із* *прізвищем* *викладача* *і* *вставляє* *у* *поле* *з* *дисциплінами* *студента*

void StudentPage::tableItemClicked(int row, int column)

{

*if*(!column)

{

QTableWidgetItem \*item = *new* QTableWidgetItem();

item->setFlags(item->flags() ^ Qt::ItemIsEditable);

QString subject = ui->comboBox\_MyDisciplines->currentText();

QString surname = ui->tableWidget\_Info->item(row,column)->text();

*for*(int i = 0; i < ui->tableWidget\_Disciplines->rowCount(); ++i)

*if*(ui->tableWidget\_Disciplines->item(i,0)->text() == subject)

{

ui->tableWidget\_Disciplines->setItem(i,column + 1, *new* QTableWidgetItem(surname));

}

ui->tableWidget\_Info->clearSelection();

}

}

*//Виводить* *інформацію* *обраних* *вчителів*

void StudentPage::PrintInfo(QTableWidget \*tableTeacherInfo, QList<Teacher> list, int mode)

{

tableTeacherInfo->setRowCount(list.size());

*for*(int i = 0; i < list.size(); ++i)

{

tableTeacherInfo->setItem(i,0,*new* QTableWidgetItem(list[i].surname));

tableTeacherInfo->setItem(i,1,*new* QTableWidgetItem(list[i].GetPosition().second));

*if*(mode == PRINT\_COURSE)

{

tableTeacherInfo->setItem(i,2,*new* QTableWidgetItem(list[i].GetDisciplines()[0].GetName().replace("\_"," ")));

tableTeacherInfo->setItem(i,3,*new* QTableWidgetItem(list[i].GetDisciplines()[0].GetCourse()));

}

*else* *if*(mode == PRINT\_DISCIPLINE\_MODE)

{

*for*(int j = 0; j < list[i].GetDisciplines().size(); ++j)

tableTeacherInfo->setItem(i,2+j,*new* QTableWidgetItem(list[i].GetDisciplines()[j].GetName().replace("\_"," ")));

}

}

}

*//Викликає* *метод* *студента* *для* *сортування* *викладачів* *за* *прізвищем* *і* *виводить* *дані*

void StudentPage::on\_pushButton\_SortSurname\_clicked()

{

ui->tableWidget\_Info->setColumnCount(2);

ui->tableWidget\_Info->setRowCount(teachersList.size());

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

student.SortTeacherListBySurname(teachersList);

PrintInfo(ui->tableWidget\_Info,teachersList,0);

}

*//Викликає* *метод* *студента* *для* *сортування* *викладачів* *за* *посадою* *і* *виводить* *дані*

void StudentPage::on\_pushButton\_SortPosition\_clicked()

{

ui->tableWidget\_Info->setColumnCount(2);

ui->tableWidget\_Info->setRowCount(teachersList.size());

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

student.SortTeacherListByPosition(teachersList);

PrintInfo(ui->tableWidget\_Info,teachersList,0);

}

*//Викликає* *метод* *студента* *для* *пошуку* *викладачів* *з* *одною* *дисципліною* *і* *виводить* *дані*

void StudentPage::on\_pushButton\_Teacher1Disc\_clicked()

{

QList<Teacher> resList;

student.FindTeachersWhoHaveOneSubject(teachersList,resList);

ui->tableWidget\_Info->setColumnCount(3);

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

ui->tableWidget\_Info->setHorizontalHeaderItem(2, *new* QTableWidgetItem("DISCIPLINE"));

PrintInfo(ui->tableWidget\_Info,resList,PRINT\_COURSE);

}

*//Викликає* *метод* *студента* *для* *пошуку* *найпопулярнішого* *викладача* *і* *виводить* *дані*

void StudentPage::on\_pushButton\_FindTheMostPopularTeacher\_clicked()

{

Teacher mostPopularTeacher;

int studentCounter;

studentCounter = student.FindTheMostPopularTeacher(teachersList,ui->comboBox\_Disciplines,mostPopularTeacher);

ui->tableWidget\_Info->setColumnCount(4);

ui->tableWidget\_Info->setRowCount(1);

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

ui->tableWidget\_Info->setHorizontalHeaderItem(2, *new* QTableWidgetItem("DISCIPLINE"));

ui->tableWidget\_Info->setHorizontalHeaderItem(3, *new* QTableWidgetItem("STUDENTS"));

ui->tableWidget\_Info->setItem(0,0,*new* QTableWidgetItem(mostPopularTeacher.surname));

ui->tableWidget\_Info->setItem(0,1,*new* QTableWidgetItem(mostPopularTeacher.GetPosition().second));

ui->tableWidget\_Info->setItem(0,2,*new* QTableWidgetItem(ui->comboBox\_Disciplines->currentText()));

ui->tableWidget\_Info->setItem(0,3,*new* QTableWidgetItem(QString::number(studentCounter)));

}

*//Виводить* *дані* *всіх* *вчителів* *котрі* *мають* *передану* *дисципліну*

void StudentPage::ShowTeachersListInfo(QString subject, QTableWidget \*table)

{

QList<Teacher> thisDiscipTeachers;

int max = teachersList[0].GetDisciplines().size();

*for*(int i = 0; i < teachersList.size(); ++i)

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

*if*(teachersList[i].GetDisciplines()[j].GetName().replace("\_"," ") == subject)

thisDiscipTeachers.push\_back(teachersList[i]);

*for*(int i = 1; i < thisDiscipTeachers.size(); ++i)

*if*(max < thisDiscipTeachers[i].GetDisciplines().size())

max = thisDiscipTeachers[i].GetDisciplines().size();

table->setColumnCount(2+max);

table->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

table->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

*for*(int i = 2; i < table->columnCount(); ++i)

table->setHorizontalHeaderItem(i, *new* QTableWidgetItem("DISCIPLINE"));

PrintInfo(table,thisDiscipTeachers,PRINT\_DISCIPLINE\_MODE);

}

void StudentPage::on\_pushButton\_ShowTeachers\_clicked()

{

ShowTeachersListInfo(ui->comboBox\_Disciplines->currentText(),ui->tableWidget\_Info);

}

void StudentPage::on\_pushButton\_ShowMyDisTeachers\_clicked()

{

ShowTeachersListInfo(ui->comboBox\_MyDisciplines->currentText(),ui->tableWidget\_Info);

}

*//Записує* *студенту* *його* *вибраного* *викладача* *на* *певну* *дисципліну*

void StudentPage::on\_pushButton\_ChooseTeacher\_clicked()

{

int row;

QString id;

QString subject = ui->comboBox\_MyDisciplines->currentText();

*if*(subject == "")

*return*;

*for*(int i = 0; i < ui->tableWidget\_Disciplines->rowCount(); ++i)

*if*(subject == ui->tableWidget\_Disciplines->item(i,0)->text())

{

row = i;

*break*;

}

*for*(int i = 0; i < teachersList.size(); ++i)

*if*(teachersList[i].surname == ui->tableWidget\_Disciplines->item(row,1)->text())

{

id = teachersList[i].GetPersonalId();

teachersList[i].SetIdOfPersonsByIndex(student.GetPersonalId(),teachersList[i].GetSizeOfIdOfPersons());

*break*;

}

student.ChooseTeacher(id);

*if*(student.GetSizeOfIdOfPersons() == ui->comboBox\_MyDisciplines->count())

{

*for*(int i = 0; i < studentsList.size(); ++i)

{

*if*(studentsList[i].surname.toLower() == student.surname.toLower() && studentsList[i].GetGroup().toUpper() == student.GetGroup().toUpper())

{

studentsList[i].SetIdOfPersons(student.GetIdOfPersons());

QMessageBox::information(*this*, "Add teachers", "Successful added your teachers");

*break*;

}

}

}

}

*//Знаходить* *вільних* *викладачів* *за* *вказаною* *датою*

void StudentPage::on\_pushButton\_FindFreeTeachersByDate\_clicked()

{

QList<Teacher> freeTeachers;

QLocale objectForDayOfWeek;

QDate selectedDate = ui->calendarWidget->selectedDate();

QString day = objectForDayOfWeek.dayName(selectedDate.dayOfWeek(),QLocale::ShortFormat);

QDate startDate,endDate;

QVector<QString> disciplineDays;

DisciplineManager \*tmpDiscManager = *new* DisciplineManager;

*for*(int i = 0; i < teachersList.size(); ++i)

{

bool flag = *true*;

*for*(int j = 0; j < teachersList[i].GetDisciplines().size(); ++j)

{

startDate = QDate::fromString(tmpDiscManager->GetStartDate(teachersList[i].GetDisciplines()[j].GetName()),"dd/MM/yyyy");

endDate = QDate::fromString(tmpDiscManager->GetEndDate(teachersList[i].GetDisciplines()[j].GetName()),"dd/MM/yyyy");

disciplineDays = tmpDiscManager->GetDaysOfDiscipline(teachersList[i].GetDisciplines()[j].GetName());

*for*(int k = 0; k < disciplineDays.size(); ++k)

{

*if*(day == disciplineDays[k] || selectedDate > endDate || selectedDate < startDate)

{

flag = *false*;

*continue*;

}

}

}

*if*(flag) freeTeachers.push\_back(teachersList[i]);

}

ui->tableWidget\_Info->setRowCount(freeTeachers.size());

int max = freeTeachers[0].GetDisciplines().size();

*for*(int i = 1; i < freeTeachers.size(); ++i)

*if*(max < freeTeachers[i].GetDisciplines().size())

max = freeTeachers[i].GetDisciplines().size();

ui->tableWidget\_Info->setColumnCount(2+max);

ui->tableWidget\_Info->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_Info->setHorizontalHeaderItem(1, *new* QTableWidgetItem("POSITION"));

*for*(int i = 2; i < ui->tableWidget\_Info->columnCount(); ++i)

ui->tableWidget\_Info->setHorizontalHeaderItem(i, *new* QTableWidgetItem("DISCIPLINE"));

PrintInfo(ui->tableWidget\_Info,freeTeachers,PRINT\_DISCIPLINE\_MODE);

}

*//Відкриває* *головне* *вікно* *програми*

void StudentPage::on\_commandLinkButton\_BackLink\_clicked()

{

hide();

MainWindow window;

window.show();

}

*//Створює* *контекстне* *меню*

void StudentPage::on\_tableWidget\_Info\_customContextMenuRequested(*const* QPoint &pos)

{

QMenu \*menu = *new* QMenu();

menu->setContextMenuPolicy(Qt::ActionsContextMenu);

QAction \*action = *new* QAction("Sort by surname");

QAction \*action2 = *new* QAction("Sort by position");

connect(action, *SIGNAL*(triggered()), *this*, *SLOT*(on\_pushButton\_SortSurname\_clicked()));

connect(action2, *SIGNAL*(triggered()), *this*, *SLOT*(on\_pushButton\_SortPosition\_clicked()));

menu->addActions({ action, action2 });

menu->exec(cursor().pos());

}

**~~~~~~~~~teacher.h~~~~~~~~~**

#ifndef TEACHER\_H

#define TEACHER\_H

#include "human.h"

#include <QTableWidget>

#include "discipline.h"

*//Клас* *Вчитель,* *який* *містить* *посаду,* *стаж* *та* *предмети,* *які* *викладає,* *а* *також* *дані* *про* *записаних* *студентів*

*class* Teacher:*public* Human

{

*public*:

Teacher();

~Teacher(){};

void SetPosition(QPair<int,QString> pos);

void SetExperience(int);

void SetDiscipline(Discipline discp);

void SetDisciplineNameByIndex(unsigned int index,QString discpName);

void SetEnrolledStudentsInfo(QPair<QString, QString> studInfo);

QPair<int,QString> GetPosition();

int GetExperience();

QVector<Discipline> GetDisciplines();

QVector<QPair<QString,QString>> GetEnrolledStudentsInfo();

void FindMyStudentsInDiscipline(QString subject);

void FindAllMyStudents();

void PrintInfoAboutStudents(QTableWidget \*table);

void SetPriorityToTeacherPosition();

bool CheckExistOfDiscipline(QString discipline);

void DeleteWrongDisciplines();

void DeleteDisciplines();

void SortMyStudentsBySurname();

void SortMyStudentsByGroup();

bool *operator*<(Teacher &t2)

{

*return* *this*->surname < t2.surname;

}

bool *operator*>(Teacher &another)

{

*return* *this*->position.first > another.position.first;

}

*private*:

QPair<int,QString> position;

int experience;

QVector<Discipline> disciplines;

QVector<QPair<QString,QString>> enrolledStudentsInfo; *//first* *-* *surname,* *second* *-* *group*

};

*extern* QList<Teacher> teachersList;

*extern* Teacher teacher;

#endif *//* *TEACHER\_H*

**~~~~~~~~~teacherMethods.cpp~~~~~~**

#include "Teacher.h"

#include "student.h"

#include "discipline.h"

#include "disciplinemanager.h"

Teacher::Teacher(){}

void Teacher::SetPosition(QPair<int,QString> pos)

{

*this*->position = pos;

}

void Teacher::SetExperience(int exp)

{

*this*->experience = exp;

}

void Teacher::SetDiscipline(Discipline discp)

{

*this*->disciplines.push\_back(discp);

}

void Teacher::SetDisciplineNameByIndex(unsigned int index,QString discpName)

{

*if*(index > *this*->disciplines.size())

*return*;

*this*->disciplines[index].SetName(discpName);

}

void Teacher::SetEnrolledStudentsInfo(QPair<QString, QString> studInfo)

{

*this*->enrolledStudentsInfo.push\_back(studInfo);

}

QPair<int,QString> Teacher::GetPosition()

{

*return* position;

}

int Teacher::GetExperience()

{

*return* experience;

}

QVector<Discipline> Teacher::GetDisciplines()

{

*return* disciplines;

}

QVector<QPair<QString,QString>> Teacher::GetEnrolledStudentsInfo()

{

*return* *this*->enrolledStudentsInfo;

}

*//Знаходить* *студентів* *викладача* *за* *переданою* *дисципліною*

void Teacher::FindMyStudentsInDiscipline(QString subject)

{

QVector<QString> tmpVect;

QPair<QString, QString> tmpPairStudInfo;

*this*->enrolledStudentsInfo.clear();

int check = 0;

*for*(int i = 0; i < *this*->disciplines.size(); ++i)

*if*(subject == *this*->disciplines[i].GetName().replace("\_"," "))

check = 1;

*if*(!check) *return*;

DisciplineManager \*tmpDiscManager = *new* DisciplineManager;

int course = tmpDiscManager->FindCourseOfSubject(subject);

*for*(int i = 0; i < studentsList.size(); ++i)

*for*(int j = 0; j < studentsList[i].GetSizeOfIdOfPersons(); ++j)

*if*(studentsList[i].GetIdOfPersonsByIndex(j) == *this*->GetPersonalId() && course == studentsList[i].GetYear())

{

tmpVect.push\_back(studentsList[i].GetPersonalId());

tmpPairStudInfo.first = studentsList[i].surname;

tmpPairStudInfo.second = studentsList[i].GetGroup();

*this*->enrolledStudentsInfo.push\_back(tmpPairStudInfo);

}

*this*->SetIdOfPersons(tmpVect);

}

*//Знаходитить* *всіх* *студентів* *викладача*

void Teacher::FindAllMyStudents()

{

QPair<QString,QString> tmpInfoAboutStudent;

*this*->ClearDataAboutIdOfPersons();

*for*(int k = 0; k < *this*->disciplines.size(); ++k)

{

int course = *this*->disciplines[k].GetCourse();

*for*(int i = 0; i < studentsList.size(); ++i)

*for*(int j = 0; j < studentsList[i].GetSizeOfIdOfPersons(); ++j)

*if*(studentsList[i].GetIdOfPersonsByIndex(j) == *this*->GetPersonalId() && course == studentsList[i].GetYear())

{

*this*->SetIdOfPersonsByIndex(studentsList[i].GetPersonalId(),*this*->GetSizeOfIdOfPersons());

tmpInfoAboutStudent.first = studentsList[i].surname;

tmpInfoAboutStudent.second = studentsList[i].GetGroup();

*this*->enrolledStudentsInfo.push\_back(tmpInfoAboutStudent);

}

}

}

*//Виводить* *інформацію* *про* *студентів* *викладача*

void Teacher::PrintInfoAboutStudents(QTableWidget \*table)

{

table->clear();

table->setColumnCount(2);

table->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

table->setHorizontalHeaderItem(1, *new* QTableWidgetItem("GROUP"));

*for*(int i = 0; i < *this*->GetSizeOfIdOfPersons(); ++i)

{

table->setItem(i,0,*new* QTableWidgetItem(*this*->enrolledStudentsInfo[i].first));

table->setItem(i,1,*new* QTableWidgetItem(*this*->enrolledStudentsInfo[i].second));

}

}

*//Встановлює* *пріоритет* *відповідно* *до* *посади* *викладача*

void Teacher::SetPriorityToTeacherPosition()

{

*if*(*this*->position.second == "Professor")

{

*this*->position.first = 4;

}

*else* *if*(*this*->position.second == "Docent")

{

*this*->position.first = 3;

}

*else* *if*(*this*->position.second == "Lecturer")

{

*this*->position.first = 2;

}

*else* *if*(*this*->position.second == "Assistant")

{

*this*->position.first = 1;

}

*else* *if*(*this*->position.second == "Laborant")

{

*this*->position.first = 0;

}

*else*

*this*->position.first = -1;

}

*//Видаляє* *некоректні* *дисципліни*

void Teacher::DeleteWrongDisciplines()

{

*for*(int i = 0; i < *this*->disciplines.size(); ++i)

{

*if*(!CheckExistOfDiscipline(*this*->disciplines[i].GetName()))

{

*if*( i == *this*->disciplines.size()-1)

*this*->disciplines.pop\_back();

*else* *if*( i == 0 )

*this*->disciplines.pop\_front();

*else*

*this*->disciplines.removeAt(i);

}

}

}

*//Перевіряє* *чи* *дана* *дисципліна* *існує* *у* *списку*

bool Teacher::CheckExistOfDiscipline(QString discipline)

{

bool check = *false*;

*for*(int j = 0; j < listOfDisciplines.size(); ++j)

*if*(discipline == listOfDisciplines[j].GetName())

check = *true*;

*return* check;

}

*//Сортує* *студентів* *за* *прізвищем(в* *алфавітному* *порядку)*

void Teacher::SortMyStudentsBySurname()

{

sort(*this*->enrolledStudentsInfo.begin(),*this*->enrolledStudentsInfo.end(),

[](QPair<QString,QString> &firStudSurname, QPair<QString,QString> &secStudSurname)

{*return* firStudSurname.first < secStudSurname.first;});

}

*//Сортує* *студентів* *за* *групою(в* *алфавітному* *порядку)*

void Teacher::SortMyStudentsByGroup()

{

sort(*this*->enrolledStudentsInfo.begin(),*this*->enrolledStudentsInfo.end(),

[](QPair<QString,QString> &firStudGroup, QPair<QString,QString> &secStudGroup)

{*return* firStudGroup.second < secStudGroup.second;});

}

*//Видаляє* *всю* *інформацію* *про* *дисципліни*

void Teacher::DeleteDisciplines()

{

*this*->disciplines.clear();

}

**~~~~~~~teacherpage.h~~~~~~~~~~**

#ifndef TEACHERPAGE\_H

#define TEACHERPAGE\_H

#include <QDialog>

*//Клас* *що* *відповідає* *за* *вікно* *вчителя*

*namespace* **Ui** {

*class* **TeacherPage**;

}

*class* **TeacherPage** : *public* QDialog

{

Q\_OBJECT

*public*:

*explicit* **TeacherPage**(QWidget \*parent = *nullptr*);

void **PrintTeacherDisciplinesInfo**();

~***TeacherPage***();

*private* slots:

void **on\_pushButton\_ShowStudents\_clicked**();

void **on\_pushButton\_AllEnrolledStud\_clicked**();

void **on\_pushButton\_SortStudentsByName\_clicked**();

void **on\_pushButton\_SortStudentsbyGroup\_clicked**();

void **on\_commandLinkButton\_BackLink\_clicked**();

*private*:

Ui::TeacherPage \*ui;

};

#endif *//* *TEACHERPAGE\_H*

**~~~~~~~~~~~teacherpage.cpp~~~~~~~~**

#include "teacherpage.h"

#include "ui\_teacherpage.h"

#include "mainwindow.h"

#include "Teacher.h"

#include "loginpage.h"

TeacherPage::TeacherPage(QWidget \*parent) :

QDialog(parent),

ui(*new* Ui::TeacherPage)

{

ui->setupUi(*this*);

QStringList parts = teacher.surname.toLower().split(' ', QString::SkipEmptyParts);

*for* (int i = 0; i < parts.size(); ++i)

parts[i].replace(0, 1, parts[i][0].toUpper());

ui->label\_Surname->setText("Welcome " + parts.join(" "));

ui->pushButton\_SortStudentsByName->hide();

ui->pushButton\_SortStudentsbyGroup->hide();

PrintTeacherDisciplinesInfo();

QPixmap pixmap("D:\\University\\Project\\PrivateSchool\\Icons\\list.ico");

QIcon ButtonIcon(pixmap);

ui->pushButton\_AllEnrolledStud->setIcon(ButtonIcon);

ui->pushButton\_AllEnrolledStud->setIconSize(pixmap.rect().size());

QPixmap studentPixmap("D:\\University\\Project\\PrivateSchool\\Icons\\student.ico");

ui->pushButton\_ShowStudents->setIcon(QIcon(studentPixmap));

ui->pushButton\_ShowStudents->setIconSize(studentPixmap.rect().size());

*if*(teacher.GetDisciplines().size() == 1)

{

ui->comboBox\_Disciplines->hide();

ui->pushButton\_AllEnrolledStud->hide();

}

*for*(int i = 0; i < teacher.GetDisciplines().size(); ++i)

ui->comboBox\_Disciplines->addItem(teacher.GetDisciplines()[i].GetName().replace("\_"," "));

}

TeacherPage::~TeacherPage()

{

*delete* ui;

}

*//Виводить* *інформацію* *про* *дисципліни* *викладача*

void TeacherPage::PrintTeacherDisciplinesInfo()

{

int maxDaysCounter = 0;

*for*(int i = 0; i < teacher.GetDisciplines().size(); ++i)

*if*(maxDaysCounter < teacher.GetDisciplines()[i].GetDaysInWeek().size())

maxDaysCounter = teacher.GetDisciplines()[i].GetDaysInWeek().size();

ui->tableWidget\_DisciplinesInfo->setColumnCount(3 + maxDaysCounter);

ui->tableWidget\_DisciplinesInfo->setRowCount(teacher.GetDisciplines().size());

ui->tableWidget\_DisciplinesInfo->setHorizontalHeaderItem(0, *new* QTableWidgetItem("DISCIPLINE"));

ui->tableWidget\_DisciplinesInfo->setHorizontalHeaderItem(1, *new* QTableWidgetItem("START PERIOD"));

ui->tableWidget\_DisciplinesInfo->setHorizontalHeaderItem(2, *new* QTableWidgetItem("END PERIOD"));

*for*(int i = 3; i < 3 + maxDaysCounter; ++i)

ui->tableWidget\_DisciplinesInfo->setHorizontalHeaderItem(i, *new* QTableWidgetItem("DAY"));

*for*(int i = 0; i < teacher.GetDisciplines().size(); ++i)

{

ui->tableWidget\_DisciplinesInfo->setItem(i,0,*new* QTableWidgetItem(teacher.GetDisciplines()[i].GetName().replace("\_"," ")));

ui->tableWidget\_DisciplinesInfo->setItem(i,1,*new* QTableWidgetItem(teacher.GetDisciplines()[i].GetStartDate()));

ui->tableWidget\_DisciplinesInfo->setItem(i,2,*new* QTableWidgetItem(teacher.GetDisciplines()[i].GetEndDate()));

*for*(int j = 3, b = 0; j < 3 + teacher.GetDisciplines()[i].GetDaysInWeek().size(); ++j, ++b)

ui->tableWidget\_DisciplinesInfo->setItem(i,j,*new* QTableWidgetItem(teacher.GetDisciplines()[i].GetDaysInWeek()[b]));

}

}

*//Виводить* *інформацію* *про* *студентів* *викладача* *за* *вибраною* *дисципліною*

void TeacherPage::on\_pushButton\_ShowStudents\_clicked()

{

ui->pushButton\_SortStudentsByName->show();

ui->pushButton\_SortStudentsbyGroup->show();

QString subject = ui->comboBox\_Disciplines->currentText();

teacher.FindMyStudentsInDiscipline(subject);

ui->tableWidget\_StudentsInfo->setRowCount(teacher.GetSizeOfIdOfPersons());

teacher.PrintInfoAboutStudents(ui->tableWidget\_StudentsInfo);

ui->label\_CountOfStudents->setText("You have " + QString::number(teacher.GetSizeOfIdOfPersons()) + " enrolled students");

}

*//Виводить* *інформацію* *про* *всіх* *студентів* *викладача*

void TeacherPage::on\_pushButton\_AllEnrolledStud\_clicked()

{

ui->pushButton\_SortStudentsByName->show();

ui->pushButton\_SortStudentsbyGroup->show();

ui->tableWidget\_StudentsInfo->setColumnCount(2);

ui->tableWidget\_StudentsInfo->setHorizontalHeaderItem(0, *new* QTableWidgetItem("SURNAME"));

ui->tableWidget\_StudentsInfo->setHorizontalHeaderItem(1, *new* QTableWidgetItem("GROUP"));

teacher.FindAllMyStudents();

ui->tableWidget\_StudentsInfo->setRowCount(teacher.GetSizeOfIdOfPersons());

teacher.PrintInfoAboutStudents(ui->tableWidget\_StudentsInfo);

ui->label\_CountOfStudents->setText("You have " + QString::number(teacher.GetSizeOfIdOfPersons()) + " enrolled students");

}

void TeacherPage::on\_pushButton\_SortStudentsByName\_clicked()

{

teacher.SortMyStudentsBySurname();

teacher.PrintInfoAboutStudents(ui->tableWidget\_StudentsInfo);

}

void TeacherPage::on\_pushButton\_SortStudentsbyGroup\_clicked()

{

teacher.SortMyStudentsByGroup();

teacher.PrintInfoAboutStudents(ui->tableWidget\_StudentsInfo);

}

*//Відкриває* *головне* *вікно* *програми*

void TeacherPage::on\_commandLinkButton\_BackLink\_clicked()

{

hide();

MainWindow window;

window.show();

}

**~~~~~~~~~main.cpp~~~~~~~~~~~~**

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QApplication>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

MainWindow w;

QLocale::setDefault(QLocale(QLocale::English, QLocale::UnitedStates));

w.setWindowTitle("IT Private School");

w.setWindowIcon(QIcon("D:\\University\\Project\\PrivateSchool\\Icons\\school.png"));

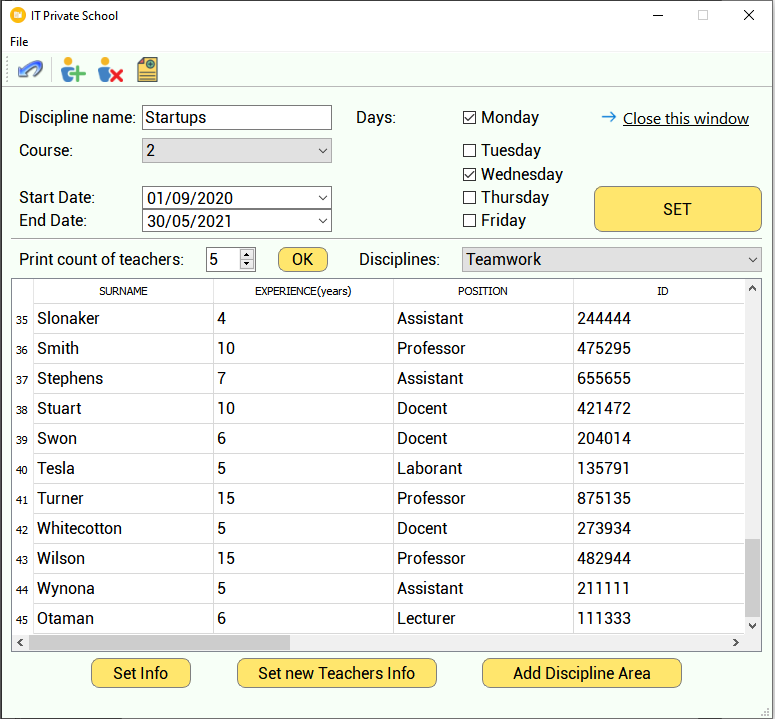
w.show();

*return* a.exec();

}

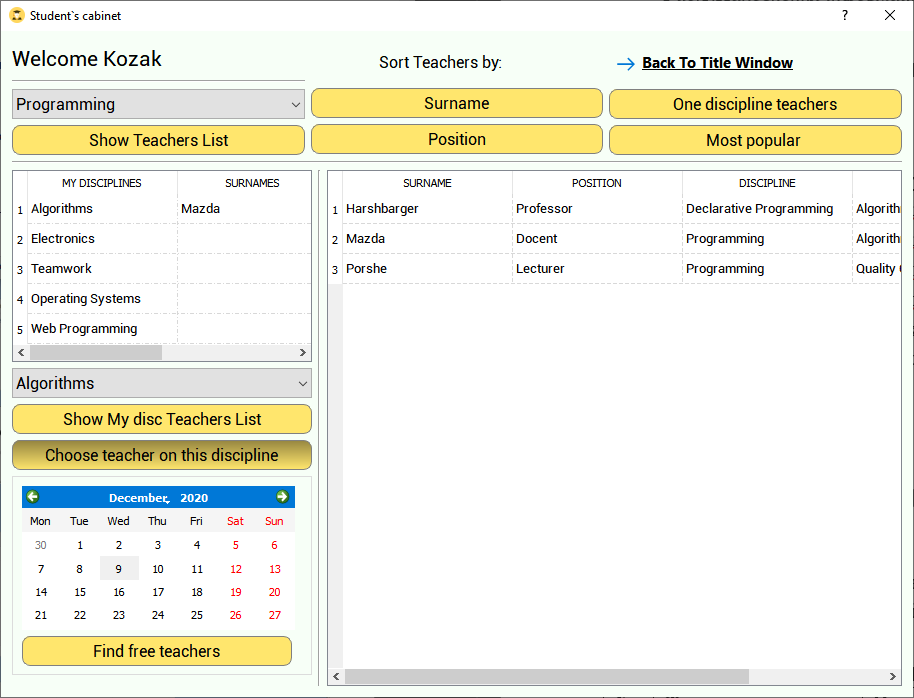
**7. Протокол роботи програми для кожного пункту завдання**

7.1. Наповнення інформацією про викладачів та дисципліни (рис.7.1).



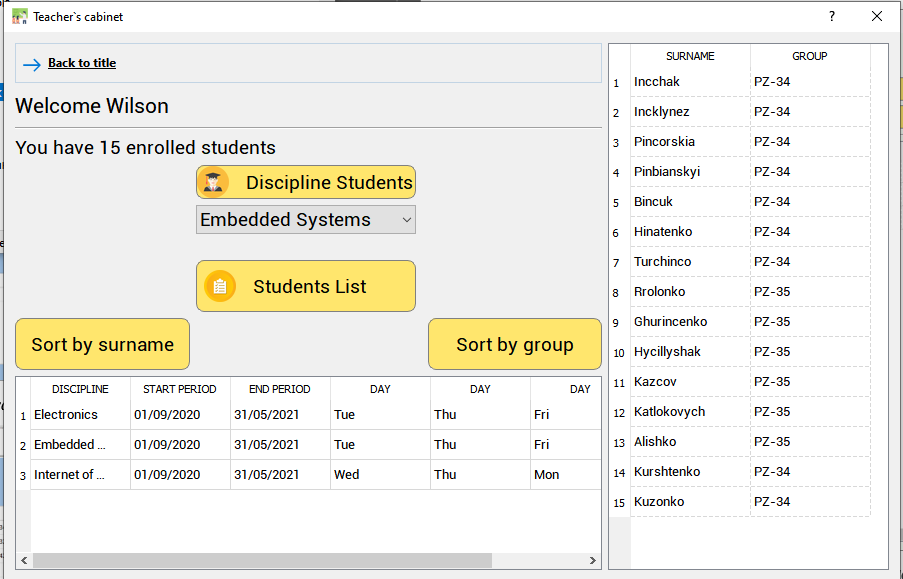
*Рис.7.1.Наповнення інформацією про викладачів та дисципліни.*

7.2. Можливість вибору викладачів за дисциплінами(рис.7.2).

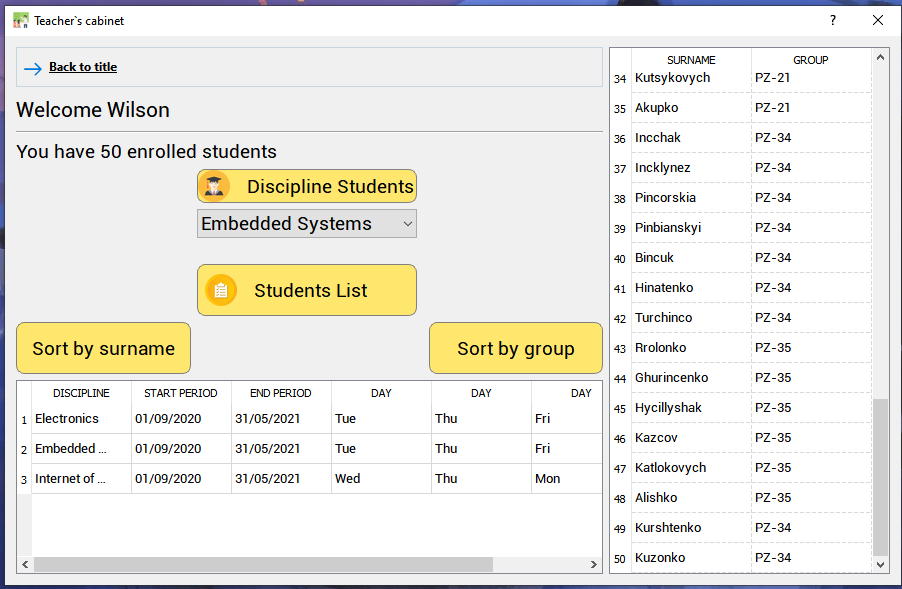


*Рис.7.2.Можливість вибору викладачів за дисциплінами.*

7.3. Можливість перегляду для викладачів студентів курсу та всіх студентів, що записалися за курсами викладача(рис.7.3,7.4).

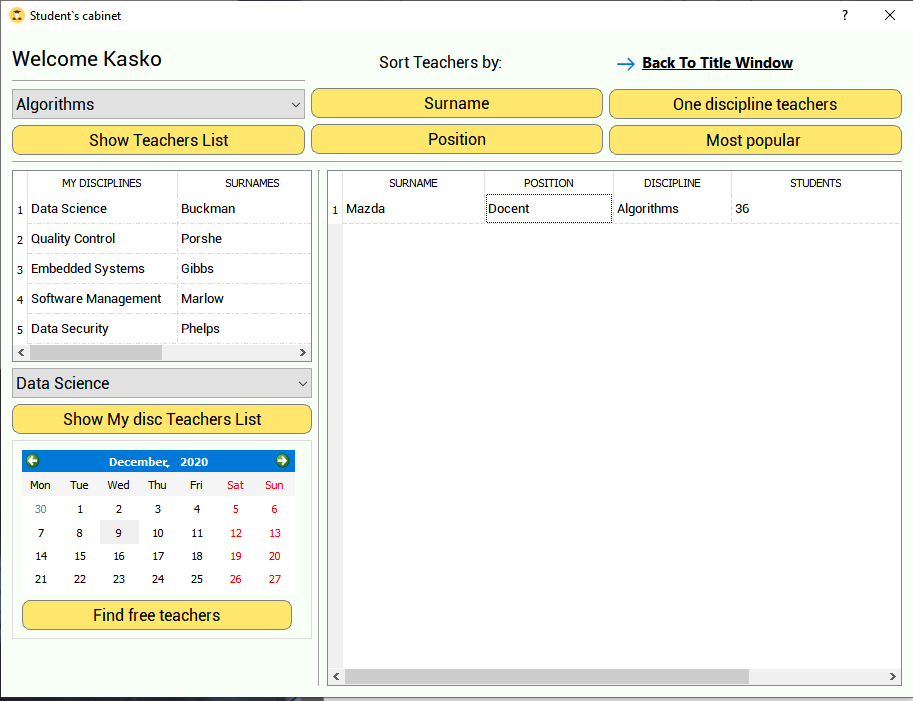


*Рис.7.3.Можливість перегляду для викладачів студентів курсу(Embedded Systems), що записалися за курсами викладача.*



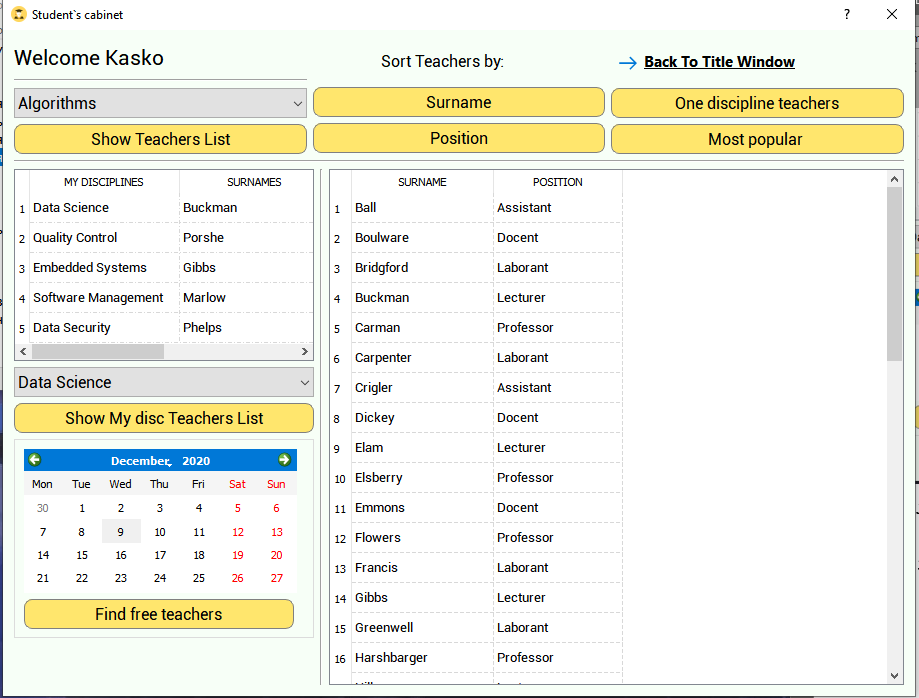
*Рис.7.4.Можливість перегляду для викладачів всіх студентів, що записалися за курсами викладача.*

7.4. Визначення найпопулярнішого викладача за заданою дисципліною(рис.7.5).

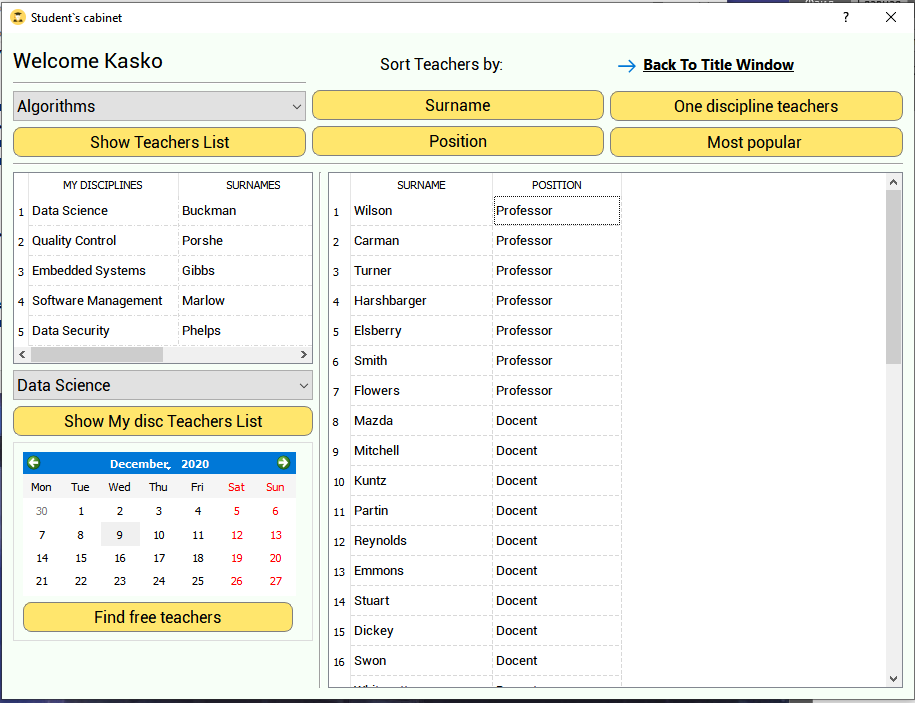


*Рис.7.5.* *Результат* *визначення найпопулярнішого викладача за заданою дисципліною(Algorithms).*

7.5. Сортування списку викладачів за посадою та прізвищем(рис.7.6,7.7).

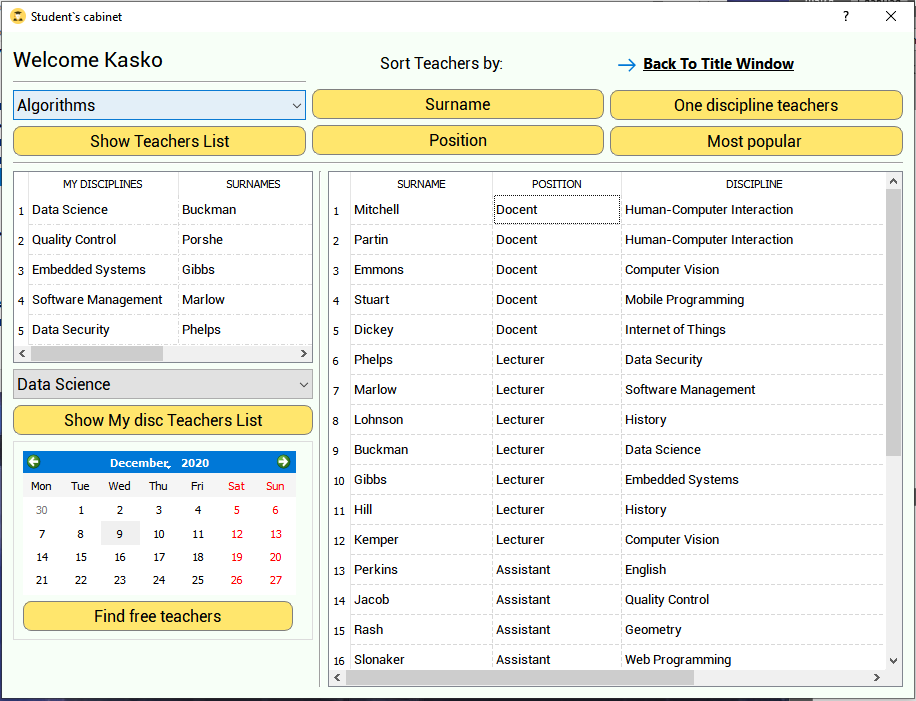


*Рис.7.6.* *Результат* *сортування списку викладачів за прізвищем.*



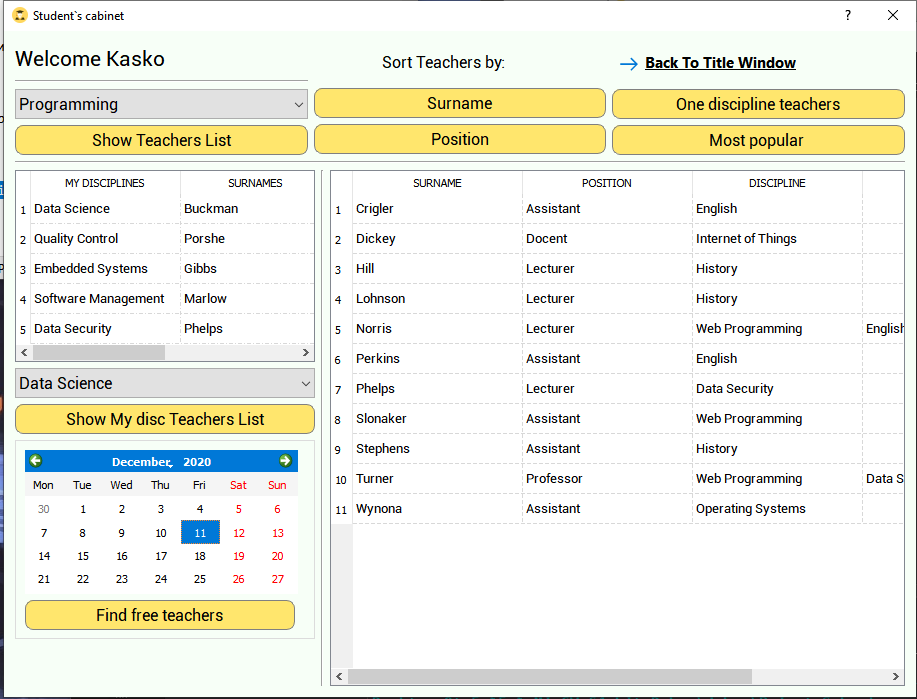
*Рис.7.7.* *Результат* *сортування списку викладачів за посадою.*

7.6. Визначити всіх викладачів, що мають тільки один курс(рис.7.8).

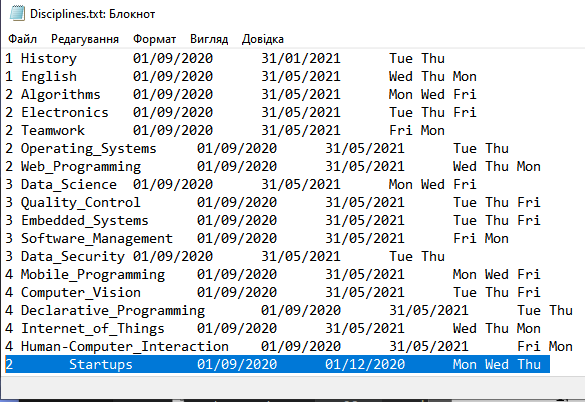


*Рис.7.8. Результат* *визначення всіх викладачів, що мають тільки один курс.*

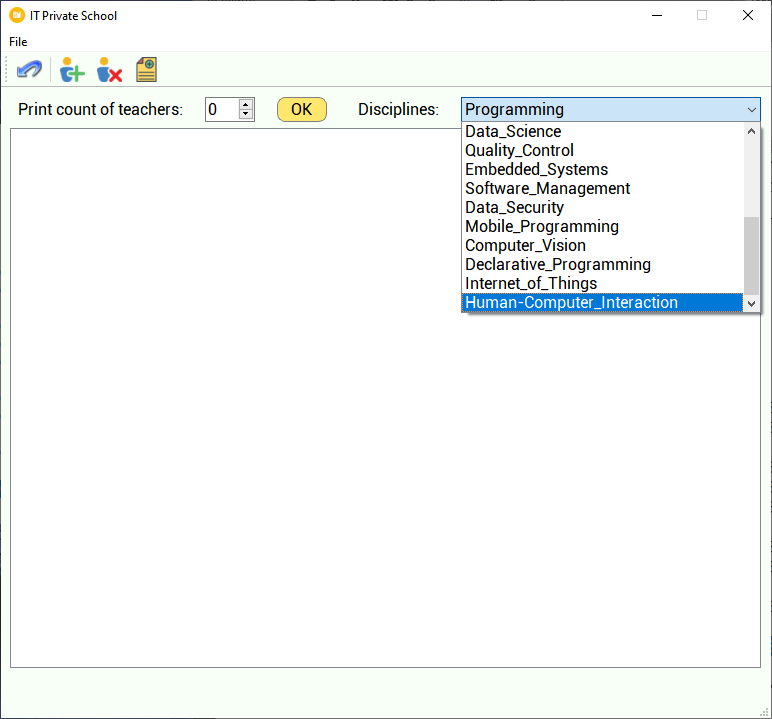
7.7. Пошук не зайнятих спеціалістів на задану дату(рис.7.9).



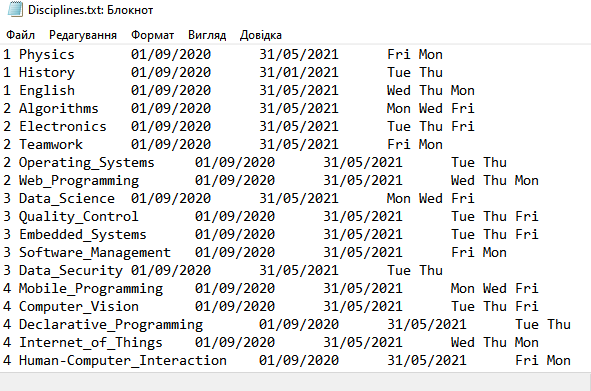
*Рис.7.9. Результат* *пошуку не зайнятих спеціалістів на задану дату.*

7.8. Автоматичне очищення історії від застарілих дат(рис.7.10-7.12).

*Рис.7.10. Інформація про дисципліну кінцева дата якої є застарілою (Число дня тестування: 09/12/2020).*



*Рис.7.11. Результат автоматичного очищення застарілих предметів (після виділеного предмета немає доданої дисципліни з файлу).*



*Рис.7.12. Результат автоматичного очищення застарілих предметів (немає доданого предмета у файлі).*

**8. Інструкція користувача та системні вимоги**

**8.1. Компоненти ПЗ**

Пакет розроблено на мові програмування С++ за стандартом С11 у середовищі розробки Qt Creator 5.15.1 (Сommunity) і може експлуатуватися під управлінням операційної системи Windows. Під час проектування підсистем відбувалося поєднання процедурно-орієнтованого підходу до програмування з об’єктно-орієнтованим. Всі функції документувались інформаційно.

Для коректної роботи пакета необхідна користувацька машина з процесором не меншим за 1 GHz, оперативною пам’яттю, не меншою за 512 Mb. Для експлуатації пакета під управлінням сімейства операційних систем Windows необхідно мати всі необхідні файли. Перелік необхідних файлів, наведено у таблиці 1.

**Набір файлів для коректної роботи програми “Private School”**

|  |  |  |  |
| --- | --- | --- | --- |
| ***№*** | ***Файл*** | ***Призначення*** | ***Належить проекту*** |
| 1 | PrivateSchool.exe | Виконавчий файл | “ **Private School** ” |
| 2 | StudentsList.txt | Текстовий файл, що містить список студентів |
| 3 | Disciplines.txt | Текстовий файл, що містить список дисциплін |
| 4 | TeachersList.txt | Текстовий файл, що містить список викладачів |

Табл.1

**8.2. Встановлення ПЗ**

Для роботи пакета запустити на виконання файл *PrivateSchool.exe*.

**8.3. Налаштування ПЗ**

Для коректної роботи програми необхідно заповнити текстові файли StudentsList.txt, Disciplines.txt, TeachersList.txt коректними даними.

**8.4. Базові функції ПЗ**

Для входу в систему як студент, вибрати у вікні пункт – «Student» (рис.8.1, номер 1).

Для входу в систему як викладач, вибрати у вікні пункт – «Teacher» (рис.8.1, номер 2).

Для реєстрації студента, вибрати у вікні пункт – «Registration» (рис.8.1, номер 3).

Для входу в систему як адміністратор, вибрати у вікні пункт – «Admin» (рис.8.1, номер 4).

Для входу в систему необхідно ввести дані, а саме прізвище та пароль (рис.8.1, номери 6, 7) та вибрати у вікні пункт – «Log In» (рис.8.1, номер 8). Для повернення з вікна входу у головне вікно необхідно вибрати пункт – «Back to title» (рис.8.2, номер 5).

У випадку входу в систему як учень, у вікні входу доступна можливість реєстрації користувача, якщо його немає у системі. Для цього необхідно вибрати пункт – «Sign up» (рис.8.2, номер 9).

Для того щоб успішно зареєструватись у системі необхідно ввести коректні дані відповідних полях (рис.8.3, номери 10,12,14,15). Поле під номером 10 на рис.8.3 має містити прізвище студента, записане англійською мовою, і перша літера має бути великою. Поле під номером 12 на рис.8.3 має містити літери з великої букви, символ ‘-’ та цифри. Після введених полів, та вибраних значень з полів 11 та 13, зображені на рис.15, необхідно вибрати пункт – «Sign up» (рис.8.3, номер 16). Для повернення до вікна входу, потрібно вибрати пункт – «Back» (рис.8.3, номер 17).

У робочому вікні студента (рис.8.4) доступні наступні функції:

Для виведення списку вчителів конкретної дисципліни, необхідно вибрати предмет у полі під номером 18 на рис.8.4 та вибрати пункт – «Show Teachers List» (рис.8.4, номер 19).Також для показу списку викладачів своїх дисциплін, потрібно вибрати предмет під номером 26, на рис. 8.4 та вибрати пункт – «Show My disc Teacher List» (рис. 8.4, номер 27).

Для сортування списку викладачів за прізвищем потрібно вибрати пункт – «Surname» (рис.8.4, номер 20).

Для сортування списку викладачів за посадою потрібно вибрати пункт – «Position» (рис.8.4, номер 21).

Для пошуку викладачів, як мають тільки одну дисципліну, потрібно вибрати пункт – «One discipline teachers» (рис.8.4, номер 23).

Для пошуку найпопулярнішого викладача (містить найбільше записаних учнів серед усіх викладачів у заданій дисципліні), потрібно вибрати пункт – «Most popular» (рис.8.4, номер 24).

Для пошуку вільних викладачів на задану дату (дата вибирається користувачем у полі під номером 28 на рис. 8.4), потрібно вибрати пункт – «Find free teachers» (рис. 8.4, номер 29).

Результат всіх вибраних пунктів відображаються на таблиці, зображена під номером 30 на рис.8.4.

Інформація про викладачів студента та його дисципліни відображаються на елементі під номером 25 на рис.8.4.

Якщо студент немає заповненої інформації про викладачів, він може обрати їх,

зробивши наступне:

1. Вибрати дисципліну на полі під номером 26 на рис.8.5.
2. Вибрати пункт – «Show My disc Teachers List»(номер 27, рис.8.5).
3. Зробити подвійний клік на елементі, де пише прізвище викладача на таблиці під номером 30 на рис.8.5.
4. Вибрати пункт – «Choose teacher on this discipline»(номер 56, рис.8.5).

У робочому вікні викладача (рис.8.6) доступні наступні функції:

Інформація про дисципліни викладача та їхні характеристики зображені на елементів під номером 35 на рис. 8.6.

Для повернення на головне вікно потрібно обрати елемент – «Back to title» під номером 31 на рис. 8.6.

Для відображення списку записаних студентів на вашу конкретну дисципліну потрібно вибрати предмет у пункті під номером 33 на рис.8.6, та обрати елемент – «Discipline Students» (рис. 8.6, номер 32).

Для відображення всіх записаних студентів потрібно обрати елемент – «Students List» (рис. 8.6, номер 34).

У робочому вікні адміністратора (рис. 8.7) доступні наступні функції:

Для відкриття файлового діалогу потрібно обрати елемент – «File» (рис. 8.7, номер 37) та натиснути на – «Open».

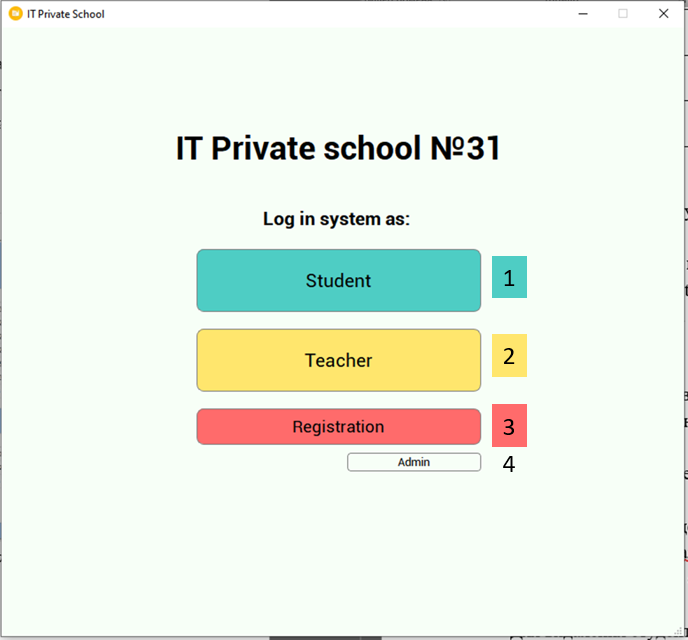
Для повернення до головного вікна потрібно обрати елемент – «Back to title»(рис.8.7, номер 37)

Для маніпуляції із вікном під номером 45, застосовуються елементи під номерами 39,40,42,43 на рис. 8.7 та 49 на рис. 8.8.

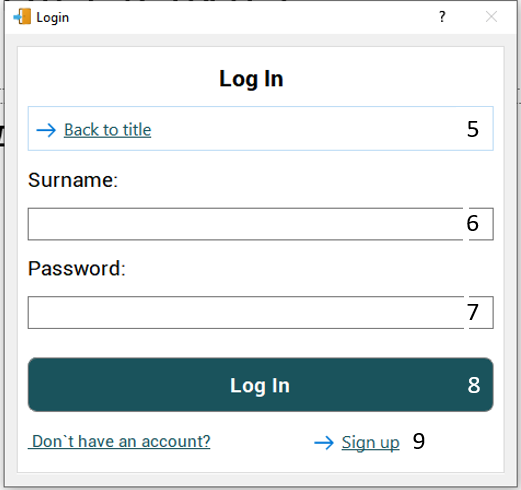
Для перегляду доступних дисциплін, обрати елемент під номером 44 на рис. 8.7.

Для зберігання інформації з таблиці обрати елемент – «SetInfo» (рис. 8.8, номер 46) та елемент – «Set new Teachers Info» (рис. 8.8, номер 47).

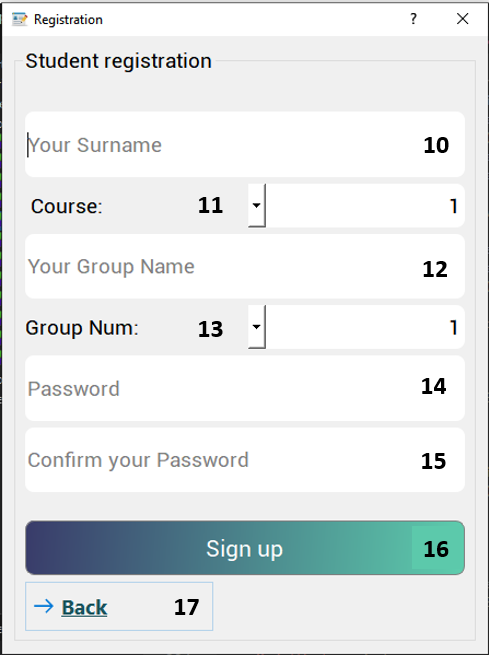
Для створення нової дисципліни обрати елемент під номером 41 на рис.8.9 і заповнити відповідні поля на рис. 8.9 (49-53) та натиснути на пункт – «SET»(рис.8.9, номер 55).



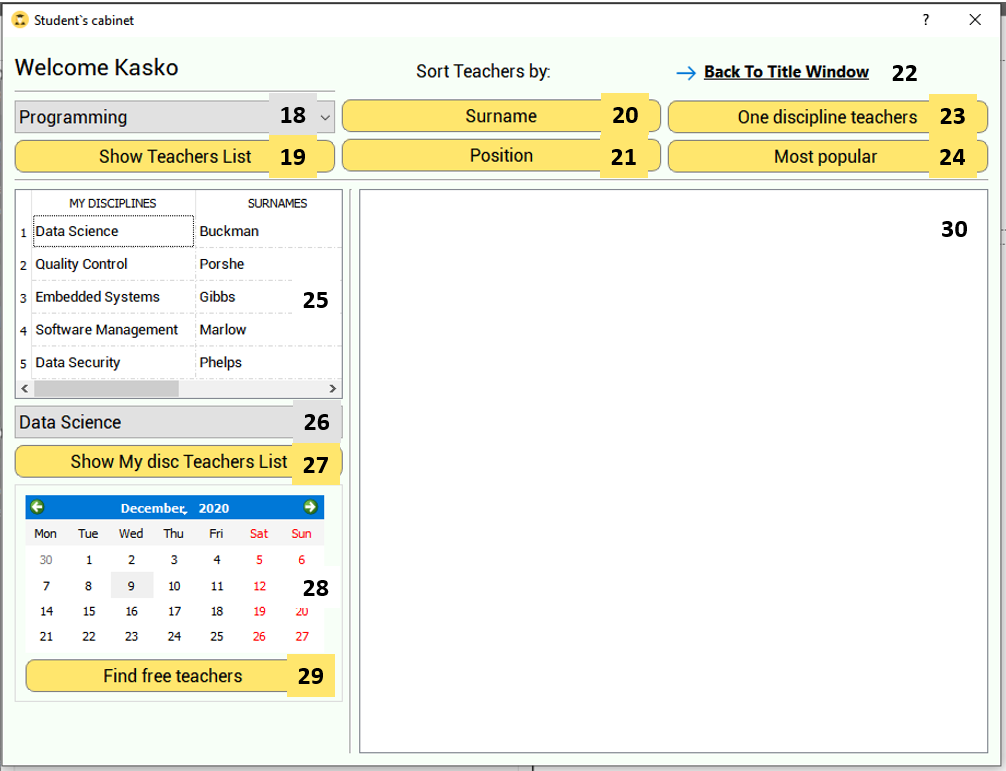
*Рис.8.1. Зображення головного вікна.*



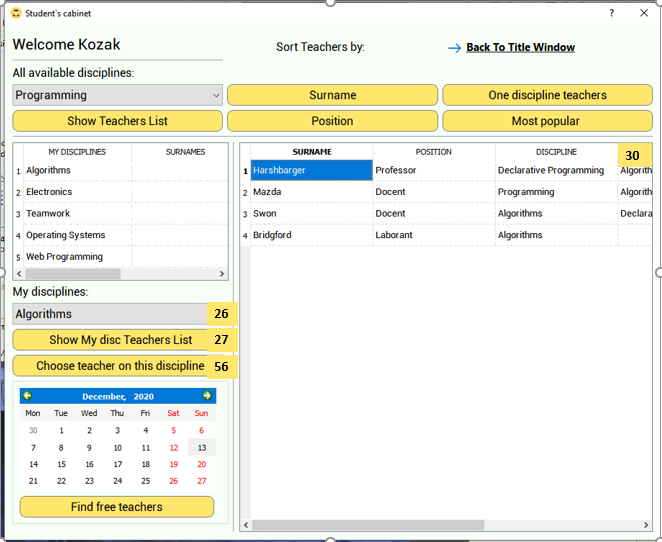
*Рис.8.2. Зображення вікна входу в систему.*



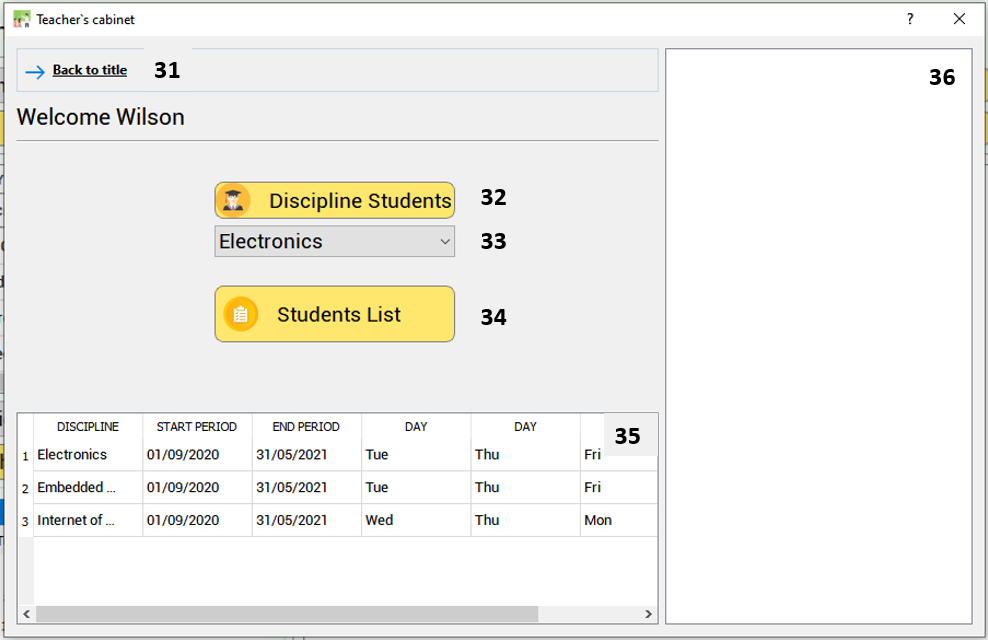
*Рис.8.3. Зображення вікна реєстрації студента.*



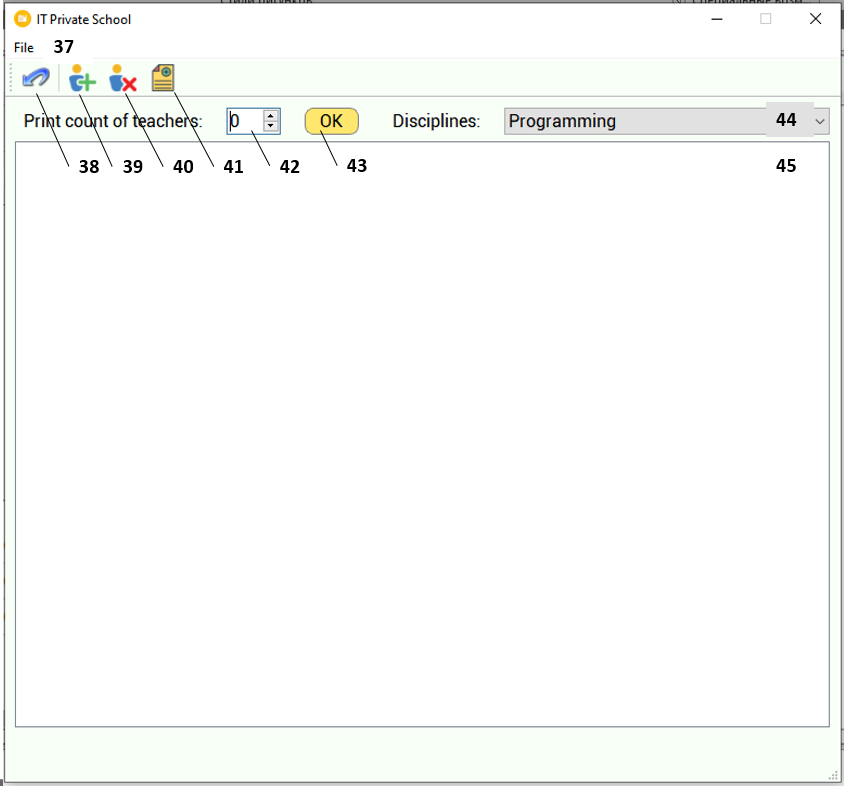
*Рис.8.4. Зображення робочого вікна студента.*



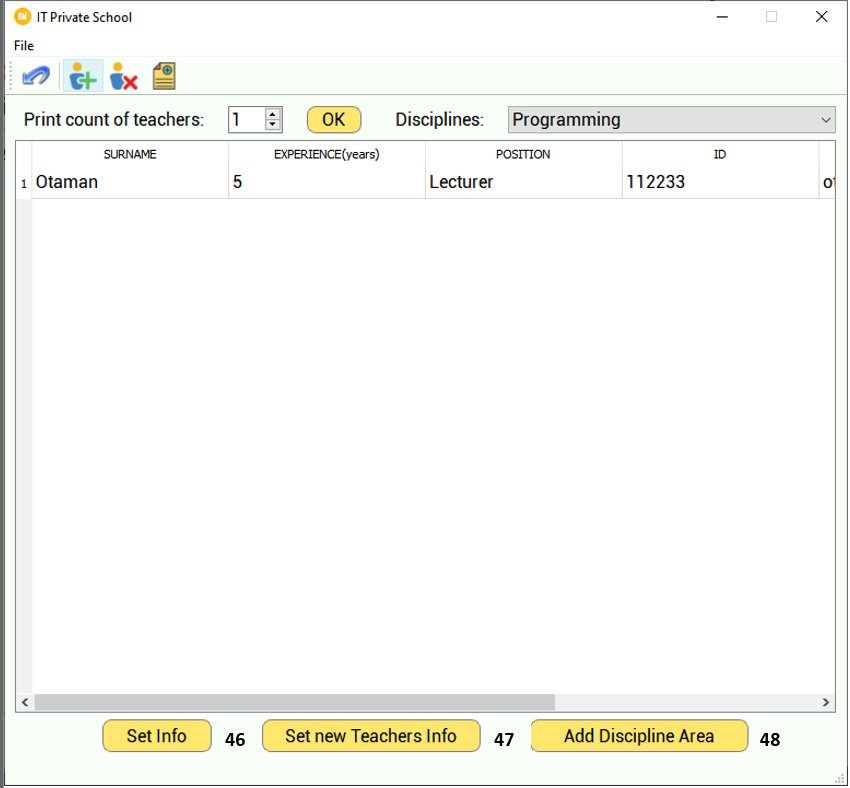
*Рис.8.5. Зображення робочого вікна студента.*



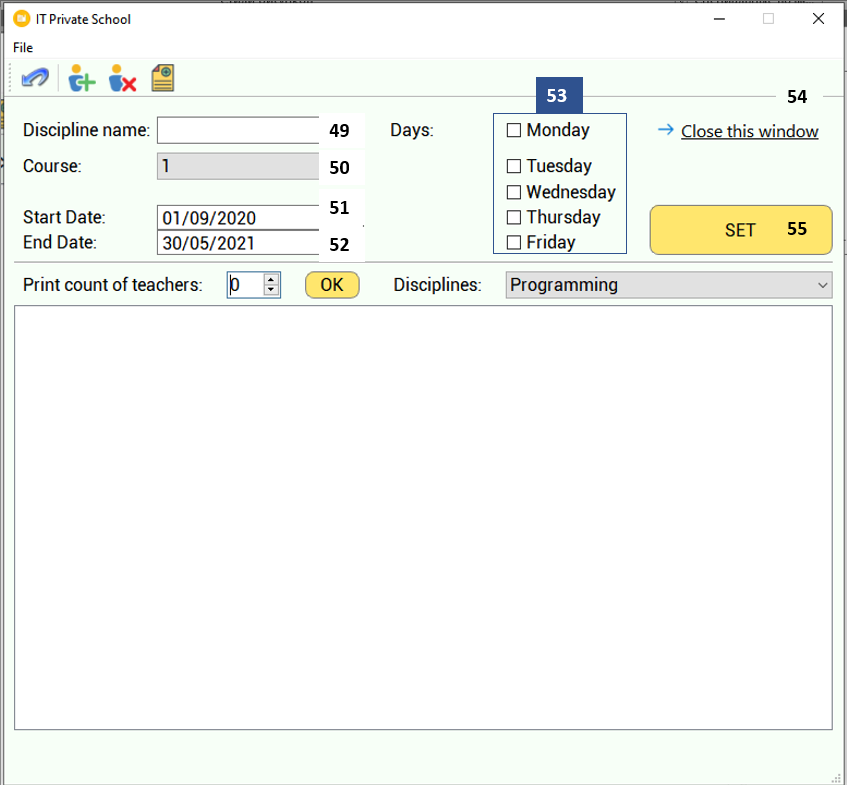
*Рис.8.6. Зображення робочого вікна викладача.*



*Рис.8.7. Зображення робочого вікна адміністратора.*



*Рис.8.8. Зображення робочого вікна адміністратора.*



*Рис.8.9. Зображення робочого вікна адміністратора.*

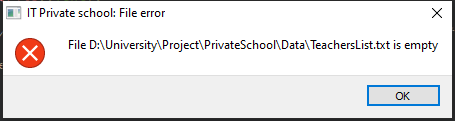
**8.5. Аналіз можливих помилок**

У випадку помилки про те, що файл порожній або його не вдалось прочитати, спробувати відкрити інший файл.

У випадку, коли файли містять некоректні дані, потрібно їх виправити та запустити програму ще раз.

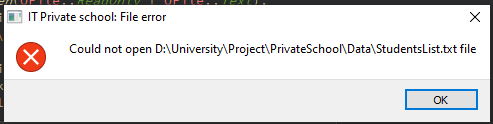
1. **Опис виняткових ситуацій**

Дане повідомлення виникає, коли заданий у програмі файл виявляється порожнім(рис.9.1).



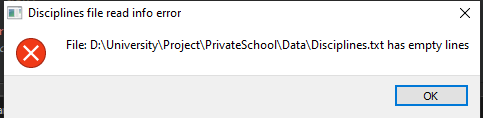
*Рис.* *9.1. Повідомлення про порожній файл.*

Дане повідомлення виникає, коли заданий у програмі файл не існує(рис. 9.2).



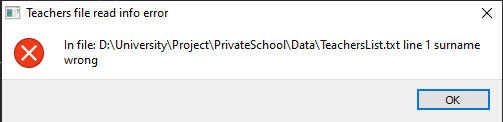
*Рис.* *9.2. Повідомлення про невдалу спробу відкриття файлу.*

Дане повідомлення виникає, коли заданий у програмі файл містить порожні рядки(рис. 9.3).



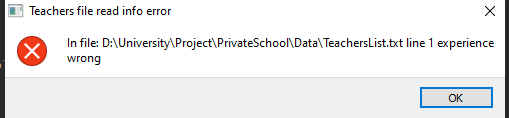
*Рис.* *9.3. Повідомлення про наявність порожніх стрічок у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці неправильний формат прізвища(рис. 9.4).



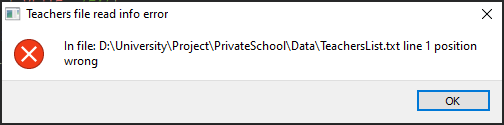
*Рис.* 9.*4. Повідомлення про некоректну інформацію прізвища у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці неправильний формат стажу(рис. 9.5).



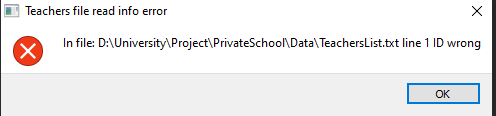
*Рис.* 9.*5. Повідомлення про некоректний стаж у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці неправильний формат посади(рис. 9.6).



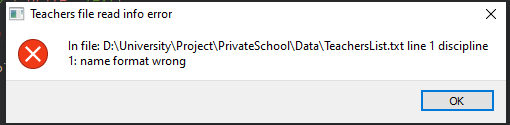
*Рис.* 9.*6. Повідомлення про некоректну посаду у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці неправильний формат ідентифікаційного номера (рис. 9.7).



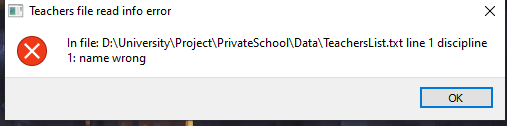
*Рис.* 9.*7. Повідомлення про некоректний ID у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці неправильний формат назви першої дисципліни (рис. 9.8).



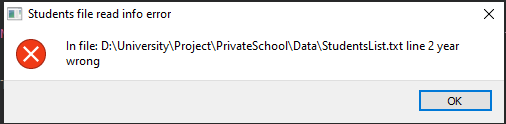
*Рис.* 9.*8. Повідомлення про некоректну назву дисципліни у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком вчителів містить у першій стрічці назву предмета, який не існує в системі (рис.29).



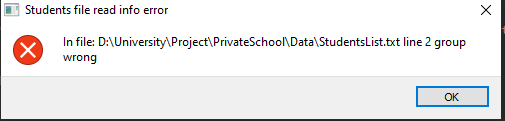
*Рис.* *9.9. Повідомлення про некоректне ім’я дисципліни у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком студентів містить у другій стрічці некоректне значення курсу (рис. 9.10).



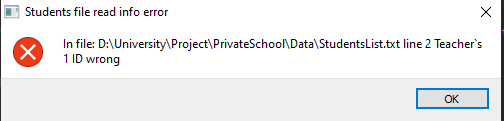
*Рис.* 9.*10. Повідомлення про некоректне значення курсу у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком студентів містить у другій стрічці некоректне значення групи (рис. 9.11).



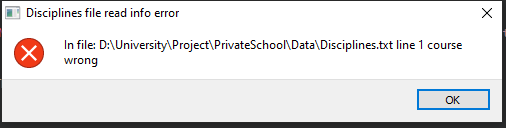
*Рис.* 9.*11. Повідомлення про некоректне значення групи у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком студентів містить у другій стрічці некоректне значення ідентифікаційного номера викладача(рис. 9.12).



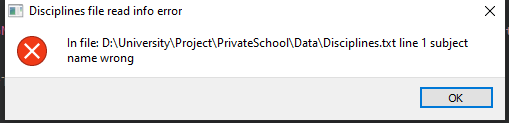
*Рис.* 9.*12. Повідомлення про некоректне значення ID викладача у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком дисциплін містить у першій стрічці некоректне значення курсу(рис. 9.13).



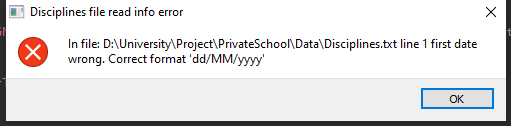
*Рис.* 9.*13. Повідомлення про некоректне значення курсу у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком дисциплін містить у першій стрічці некоректне значення імені(рис. 9.14).



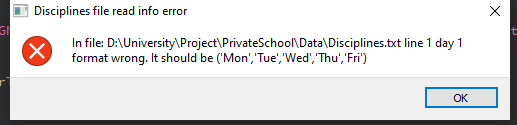
*Рис.* *9*.*14. Повідомлення про некоректне значення імені у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком дисциплін містить у першій стрічці некоректне значення початкового терміну викладання предмету(рис. 9.15).



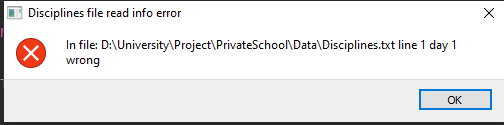
*Рис.* 9.*15. Повідомлення про некоректне значення початкового терміну у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком дисциплін містить у першій стрічці некоректне значення дня викладання предмету(рис. 9.16).



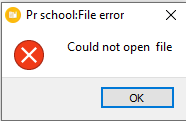
*Рис.* 9.*16. Повідомлення про некоректне значення дня у файлі.*

Дане повідомлення виникає, коли заданий у програмі файл із списком дисциплін містить у першій стрічці некоректний формат дня викладання предмету(рис. 9.17).



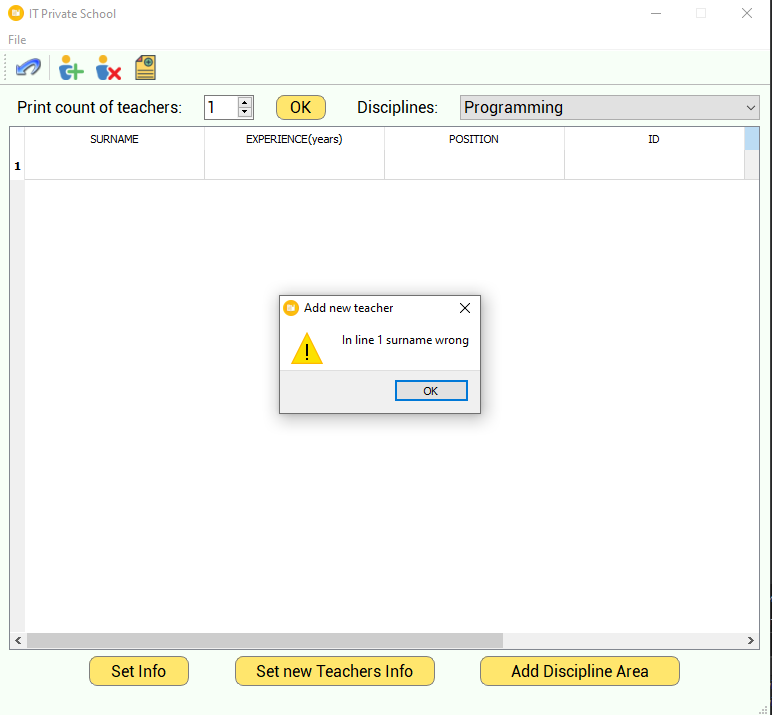
*Рис.* 9.*17. Повідомлення про некоректний формат дня у файлі.*

Дане повідомлення виникає, коли вибраний користувачем файл не вдалось відкрити (рис. 9.18).



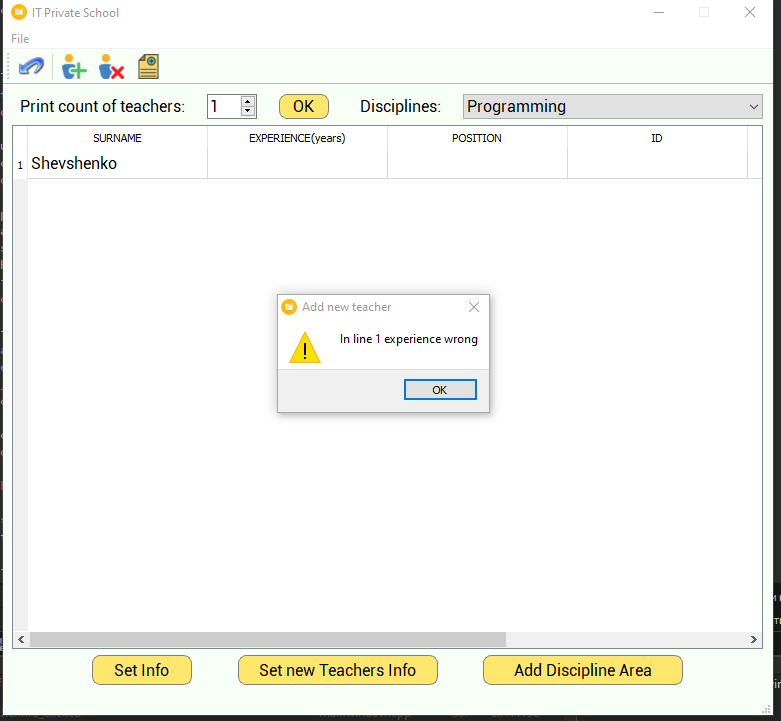
*Рис.* 9.*18. Повідомлення про некоректне відкриття файлу.*

Дане повідомлення виникає, коли користувач ввів некоректні дані прізвища у таблиці, або не ввів жодних(рис. 9.19).



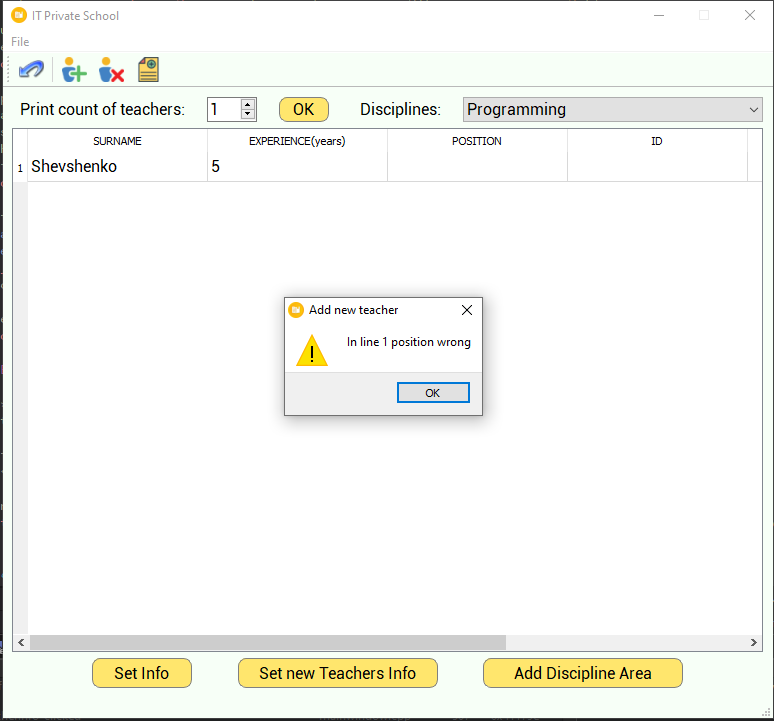
*Рис.* 9.*19. Повідомлення про некоректність введеного прізвища у таблиці.*

Дане повідомлення виникає, коли користувач ввів некоректні дані стажу в таблиці, або не ввів жодних(рис. 9.20).



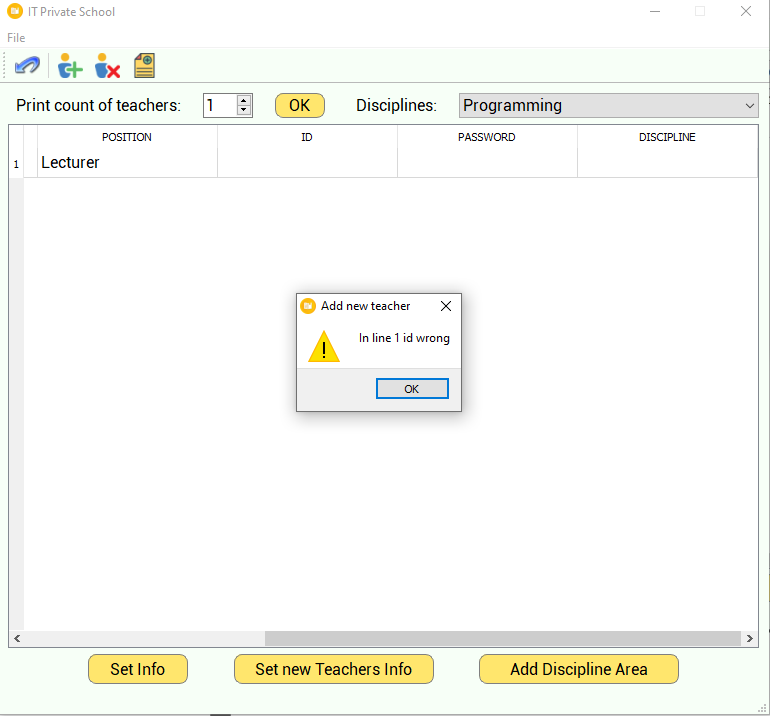
*Рис.* 9.*20. Повідомлення про некоректність введеного стажу у таблиці.*

Дане повідомлення виникає, коли користувач ввів некоректні дані посади в таблиці, або не ввів жодних(рис. 9.21).



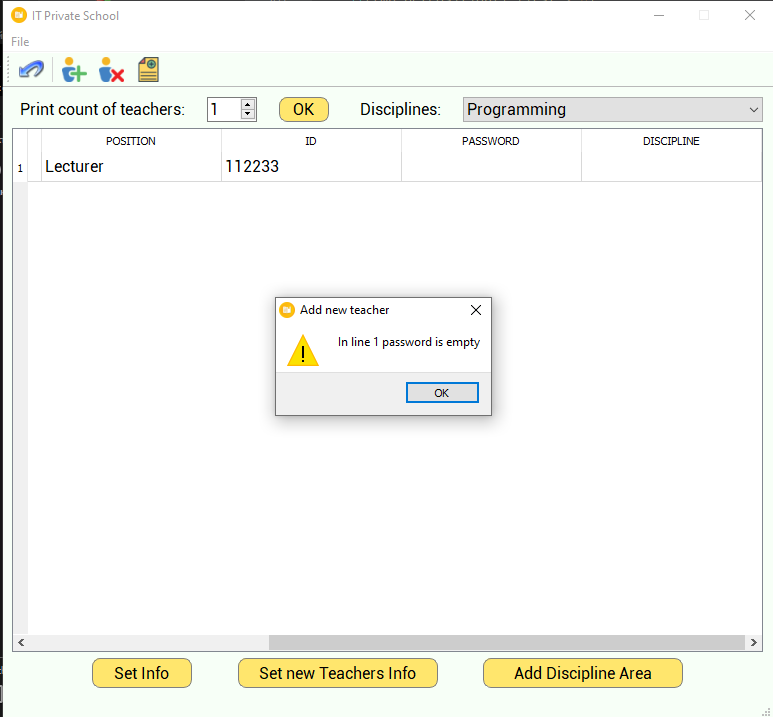
*Рис.* 9.*21. Повідомлення про некоректність введеної посади у таблиці.*

Дане повідомлення виникає, коли користувач ввів некоректні дані ID в таблиці, або не ввів жодних(рис. 9.22).



*Рис.* 9.*22. Повідомлення про некоректність введеного ID у таблиці.*

Дане повідомлення виникає, коли користувач не ввів значення паролю(рис. 9.23).



*Рис.* 9.*23. Повідомлення про некоректність введеного паролю у таблиці.*

1. **Структура файлу вхідних даних**

У програмі передбачене зчитування інформації із конкретних файлів, а саме:

* Disciplines.txt (відповідає за список всіх дисциплін)
* TeachersList.txt (відповідає за список всіх вчителів)
* StudentsList.txt (відповідає за список всіх студентів)

У Disciplines.txt дані повинні подаватись у вигляді:

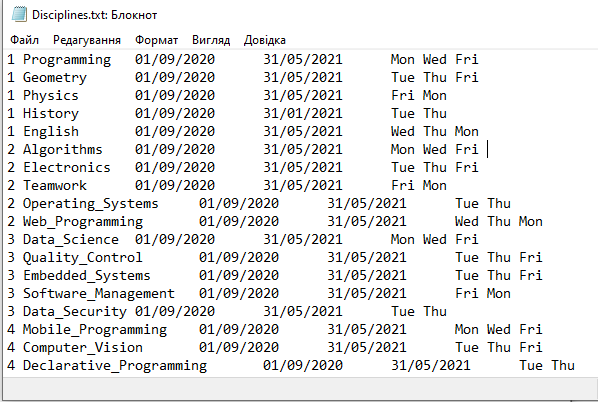
<**Курс викладання дисципліни**> <**Назва дисципліни**> <**Початковий термін викладання**> <**Кінцевий термін викладання**> <**Дні викладання**>

<Курс викладання дисципліни> вводиться число від 1 до 4 включно.

<Назва дисципліни> вводиться ім’я англійською мовою. Якщо назва складається більш ніж з одного слова, тоді добавляти символ ‘\_’ між словами. Допускається дефіс у назві дисципліни.

<Початковий термін викладання> та <Кінцевий термін викладання> записуються у форматі ***день/місяць/рік***.

<Дні викладання> записуються у форматі назви дня тижня із трьох букв англійською. Допустимі дні: **Mon Tue Wed Thu Fri**. Якщо предмет викладається у різні дні, то записати ці дні через пробіл. На рис.10.1 зображено приклад текстового файлу із правильним форматом списку дисциплін.



*Рис.10.1. Приклад текстового файлу для зчитування списку дисциплін.*

У StudentsList.txt дані повинні подаватись у вигляді:

<**Прізвище**> <**Курс навчання**> <**Група**> <**ID**> <**Пароль**> <**Список ID вчителів**>

<Прізвище> вводиться англійською мовою, бажано першу літеру з великої букви писати.

<Курс навчання> вводиться число від 1 до 4 включно.

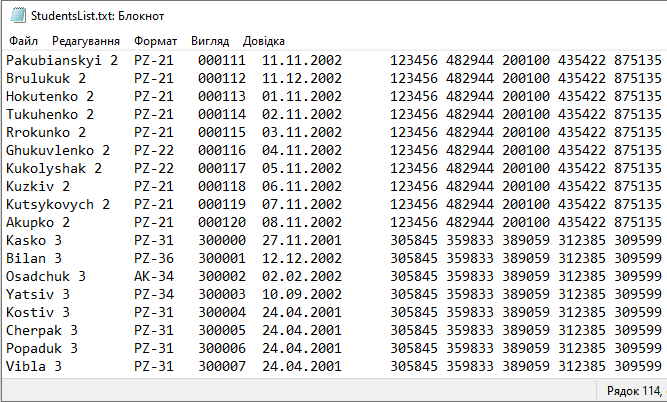
<Група> вводиться із літер англійського алфавіту, символу ‘-’ та чисел.

<ID> - ідентифікаційний номер студента. Вводиться як число.

<Пароль> дозволяється вводити як слова, так і числа та символи.

<Список ID вчителів> - ідентифікаційний номер вчителя вводиться числом відповідним до персонального ID уже існуючого вчителя у системі. Список цих номерів записується через пробіл.

На рис.10.2 зображено приклад текстового файлу із правильним форматом списку учнів.



*Рис.10.2. Приклад текстового файлу для зчитування списку учнів.*

У TeachersList.txt дані повинні подаватись у вигляді:

**<Прізвище> <Стаж> <Посада> <Персональний ID> <Пароль> <Список дисциплін>**

<Прізвище> вводиться англійською мовою, бажано першу літеру з великої букви писати.

<Стаж> вводиться числом більшим за 0.

<Посада> записується англійською мовою, і містить посаду із допустимих.

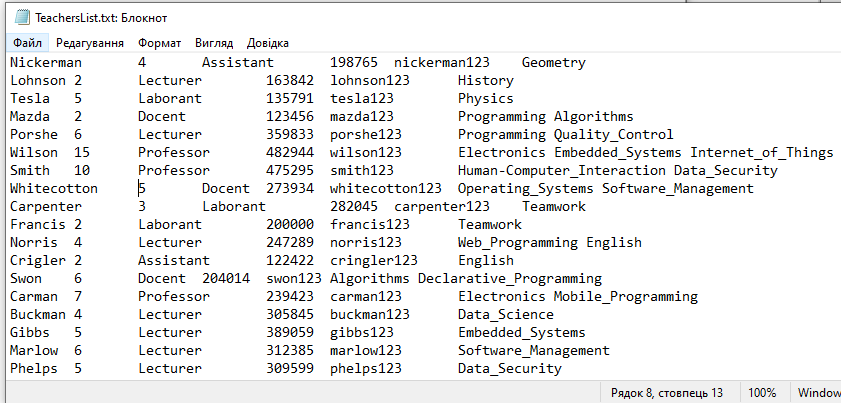
Допустимі посади: **Laborant** **Assistant Lecturer Docent Professor**.

<Персональний ID> - ідентифікаційний номер викладача. Вводиться як число.

<Пароль> дозволяється вводити як слова, так і числа та символи.

<Список дисциплін> вводяться допустимі дисципліни (записані у файлі Disciplines.txt). Список цих дисциплін записується через пробіл.

На рис.10.3 зображено приклад текстового файлу із правильним форматом списку вчителів.



*Рис.10.3. Приклад текстового файлу для зчитування списку вчителів.*

**Висновки**

На даній курсовій роботі, я реалізував програмне забезпечення, що відтворює модель навчання в приватній школі. Даний програмний застосунок був написаний на мові С++ з використанням стандартної бібліотеки шаблонів (STL) та з використанням інструмента швидкої розробки додатків Qt Creator. При проектуванні даної програми я використав об’єктно-орієнтований підхід. На практиці застосував знання про життєвий цикл програмного забезпечення.

**Список використаної літератури**

1. Бьярне Срауструп. Язык программирование С++. Краткий курс, второе издание; [И. В. Красикова] / Бьярне Страуструп. Москва: “Диалектика”, 2019. — 320 с.
2. Левус Є. В. Життєвий цикл програмного забезпечення / Є. В . Левус , Т. А. Марусенкова, О. О. Нитребич Львів : Видавництво Львівської політехніки, 2017. 208 с.
3. Прата С++. Язык программирования С++. Лекции и упражнения, 6-е издание; [пер. з англ. Ю.И. Корниенко] / Стивен Прата. Москва: “И.Д. Вильямс”, 2012. — 1248 с.
4. C++ Core Guidelines. – [Електронний ресурс] – Режим доступу по ресурсу: https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines
5. Vectors in C++ STL. – [Електронний ресурс] – Режим доступу по ресурсу: https://en.cppreference.com/w/cpp/container/vector