НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КУРСОВА РОБОТА

з дисципліни «Інженерія програмного забезпечення»
на тему: «Аналіз успішності студентів. Функціонал візуалізації даних»

Студента 2 курсу групи ІО-61
напряму підготовки: 123 - Комп'ютерна інженерія
номер залікової 6103
Валько Антон Вікторович

Керівник старший викладач, к.т.н., с.н.с.
Антонюк Андрій Іванович

Національна оцінка _____
Кількість балів _____

Науковий керівник _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Члени комісії _____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

_____ _____
(підпис) (вчене звання, науковий ступінь, прізвище та ініціали)

Київ - 2018 рік

ЗМІСТ

ВСТУП

Об’єктом розробки даної курсової роботи є програма з функціональністю аналізу успішності студентів учбового закладу.

Метою курсової роботи є закріплення теоретичних знань і практичних навичок з проектування, моделювання, розробки та тестування програмного забезпечення з графічним інтерфейсом.

Результат курсової роботи – готовий програмний додаток з графічним інтерфейсом, написаний на мові програмування Python.

Графічний інтерфейс користувача – інтерфейс між комп’ютером та його користувачем, реалізований за допомогою бібліотеки Tkinter.

1. ОГЛЯД MVC

1.1 Загальна характеристика

Model – view – controller (MVC, «модель – представлення – поведінка») – схема використання кількох шаблонів проектування, за допомогою яких модель даних програми, інтерфейс користувача і взаємодія з користувачем розділені на три окремих компонента таким чином, щоб модифікація одного з компонентів впливала на інші мінімально. Дана схема проектування часто використовується для побудови архітектурного каркаса, коли переходять від теорії до реалізації в конкретній предметній області.

Мета шаблону — гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші зміни чи розширення програм, а також надавати можливість повторного використання окремих компонентів програми. Крім того використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш зрозумілими за рахунок зменшення складності.

1.2 Структура MVC

У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних і забезпечення інтерфейсу до них. Вигляд (View) відповідальний за представлення цих даних користувачеві. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель.

- Модель (Model) є центральним компонентом шаблону MVC і відображає поведінку застосунку, незалежну від інтерфейсу користувача. Модель стосується прямого керування даними, логікою та правилами застосунку.
- Вигляд (View) може являти собою будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграму. Одночасно можуть співіснувати кілька виглядів (представлень) однієї і тієї ж інформації, наприклад гістограма для керівництва компанії й таблиці для бухгалтерії.
- Контролер (Controller) одержує вхідні дані й перетворює їх на команди для моделі чи вигляду.

Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних. Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані.

У функції контролера входить відстеження визначених подій, що виникають в результаті дій користувача. Контролер дозволяє структурувати код шляхом групування пов'язаних дій в окремий клас. Наприклад у типовому MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля.

1.3 Шаблони програмування, що використовуються в MVC

Для реалізації схеми MVC використовується досить велика кількість шаблонів проектування (залежно від складності архітектурного рішення), основні з яких «спостерігач», «стратегія» та «компонувальник».

Найбільш типова реалізація відокремлює представлення від моделі шляхом встановлення між ними протоколу взаємодії, використовуючи апарат подій

(підписка/сповіщення). При кожній зміні внутрішніх даних в моделі вона оповіщає все залежні від неї представлення, і вони оновлюються. Для цього використовується шаблон «спостерігач» При обробці реакції користувача представлення вибирає, залежно від потрібної реакції, потрібний контролер, який забезпечить той чи інший зв'язок з моделлю. Для цього використовується шаблон «стратегія», або замість цього може бути модифікація з використанням шаблону «команда». Крім того, можуть використовуватися й інші шаблони проектування, наприклад, «фабричний метод», який дозволить задати за замовчуванням тип контролера для відповідного представлення.

1.4 Завдання MVC

Основна мета застосування цієї концепції полягає в відокремленні бізнес-логіки (моделі) від її візуалізації (представлення, виду). За рахунок такого поділу підвищується можливість повторного використання. Найбільш корисне застосування даної концепції в таких випадках, коли користувач повинен бачити ті ж самі дані одночасно в різних контекстах та/або з різних точок зору. Зокрема, виконуються наступні завдання:

- До однієї моделі можна приєднати кілька представлень, при цьому не зачіпаючи реалізацію моделі. Наприклад, деякі дані можуть бути одночасно представлені у вигляді електронної таблиці, гістограми і діаграми.
- Не торкаючись реалізації представлень, можна змінити реакції на дії користувача (натискання на кнопку, введення даних), для цього досить використовувати інший контролер.
- Ряд розробників спеціалізується тільки в одній з областей: або розробляють графічний інтерфейс, або розробляють бізнес-логіку. Тому можна досягти того, що

програміст, які займаються розробкою бізнес-логіки (моделі), взагалі не будуть знати про те, який вигляд буде використовуватися.

1.5 Python

Дану курсову роботу буде виконано, використовуючи мову програмування Python та бібліотеку Tkinter. Бібліотека Tkinter зручна та має велику кількість можливостей. Вона зможе надати нам змогу управляти графічними виглядом елементів.

1.6 Використання концепції MVC в даній роботі

В даній роботі буде використовуватися модифікована версія шаблону MVC, яка об’єднує представлення і контролер в один логічний об’єкт. MVC розділить систему на дві умовні частини, а саме: модель даних (база даних або файл), вигляд даних (графічний інтерфейс користувача), об’єднаний з керуванням (логіка програми). Це буде зроблено з метою відокремлення моделі даних від інтерфейсу користувача, щоб зміни в будь-якій з цих частин системи мінімально впливали на інші частину нашої системи.

2. ПРОЕКТУВАННЯ ПРОГРАМНОГО ДОДАТКУ

2.1 Прецеденти

Можливості користувача додатку можна побачити на Рис. 2.1.

Користувач може додавати студентів до бази даних, видаляти студентів з бази даних, оновлювати відображення, переглядати відображення.

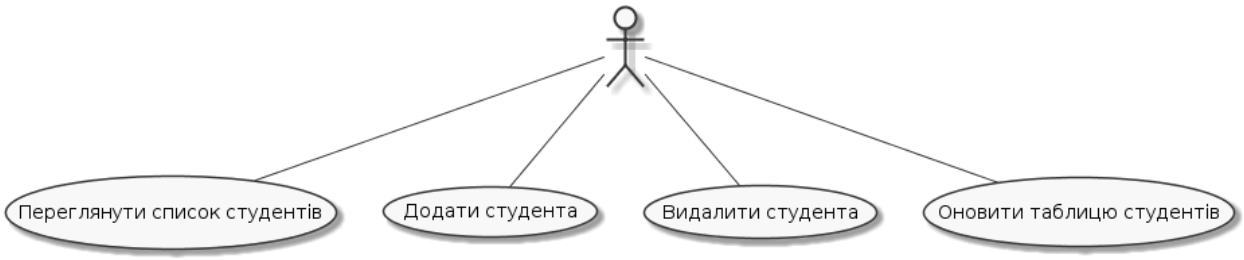


Рис. 2.1. Можливості користувача

Користувач може побачити успішність групи, вибравши курс, спеціальність та групу та натиснувши на кнопку «Вибрати» (Рис. 2.2).

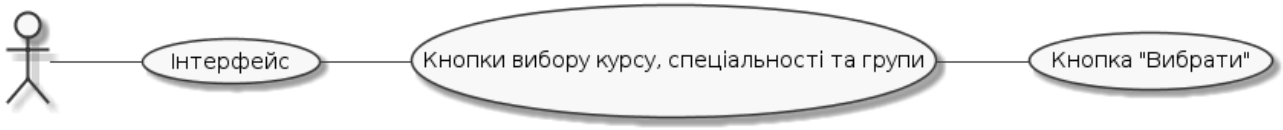


Рис. 2.2. Операція входу до вікна перегляду таблиці успішності студентів

Користувач може побачити успішність групи у вигляді діаграми, натиснувши на кнопку «Статистика» (Рис. 2.3).

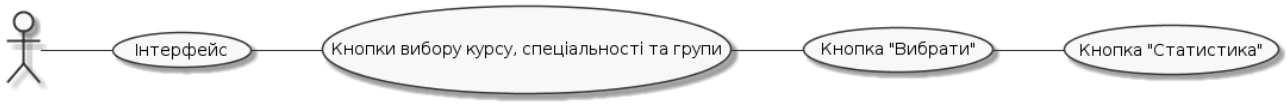


Рис. 2.3. Операція перегляду успішності групи у вигляді діаграми

Користувач може додавати та видаляти студентів з групи, увійшовши в меню редагування, натиснувши кнопку «Редагувати» (Рис. 2.4).

Рис. 2.4. Операція входу до меню редагування

Користувач може додати вказаного студента до групи, натиснувши кнопку «Додати» у вікні редагування та вписавши потрібну інформацію (Рис. 2.5).

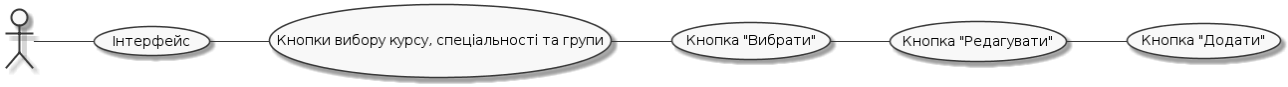


Рис. 2.5. Операція додавання студента до групи

Користувач може видалити вказаного студента з групи, натиснувши кнопку «Видалити» у вікні редагування (Рис. 2.6).

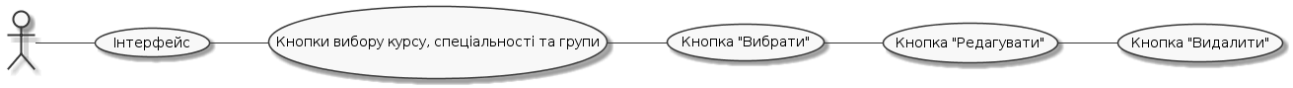


Рис. 2.6. Операція видалення студента з групи

Користувач може оновити список студентів після внесених змін, натиснувши кнопку «Оновити» (Рис. 2.7).

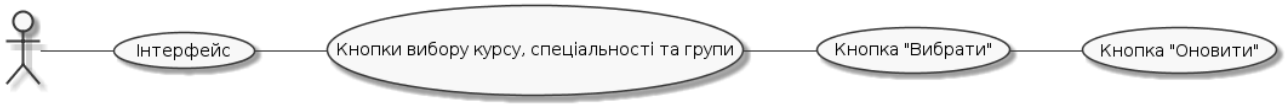


Рис. 2.7. Операція оновлення відображення таблиці студентів

2.2 Ескізи графічного інтерфейсу

На рисунках 2.8, 2.9, 2.10 та 2.11 можна побачити ескізи всіх можливих вікон програми.

Рис. 2.8 Ескіз вікна вибору групи

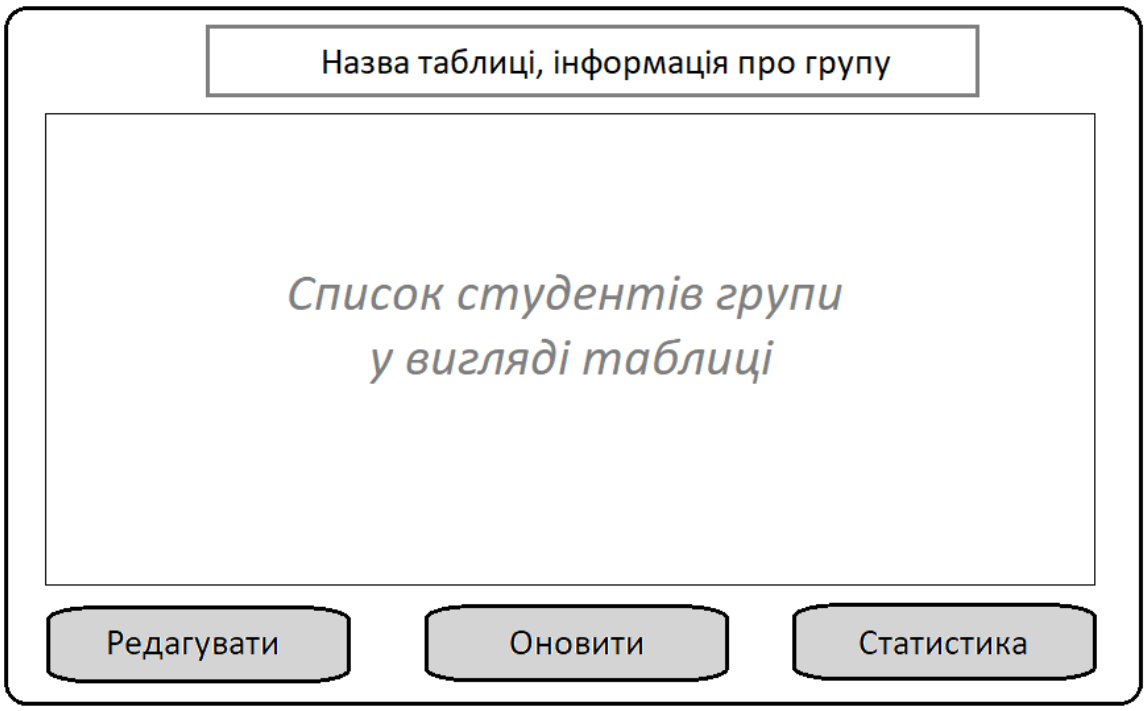


Рис. 2.9 Ескіз вікна відображення списку успішності студентів групи

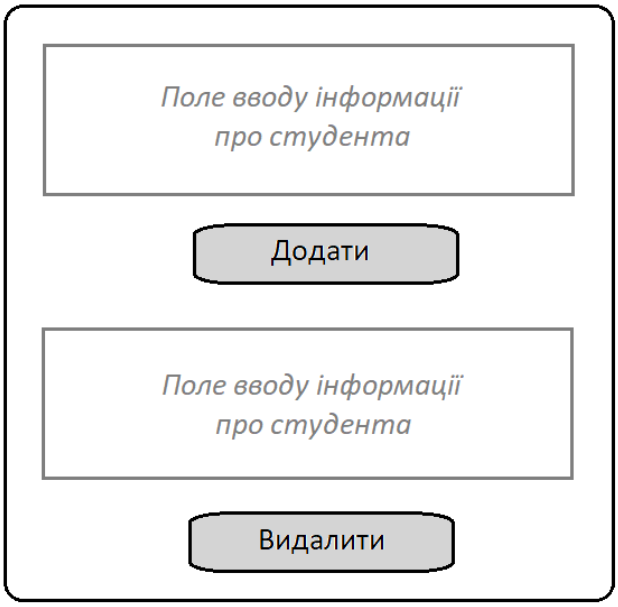


Рис. 2.10 Ескіз вікна редагування

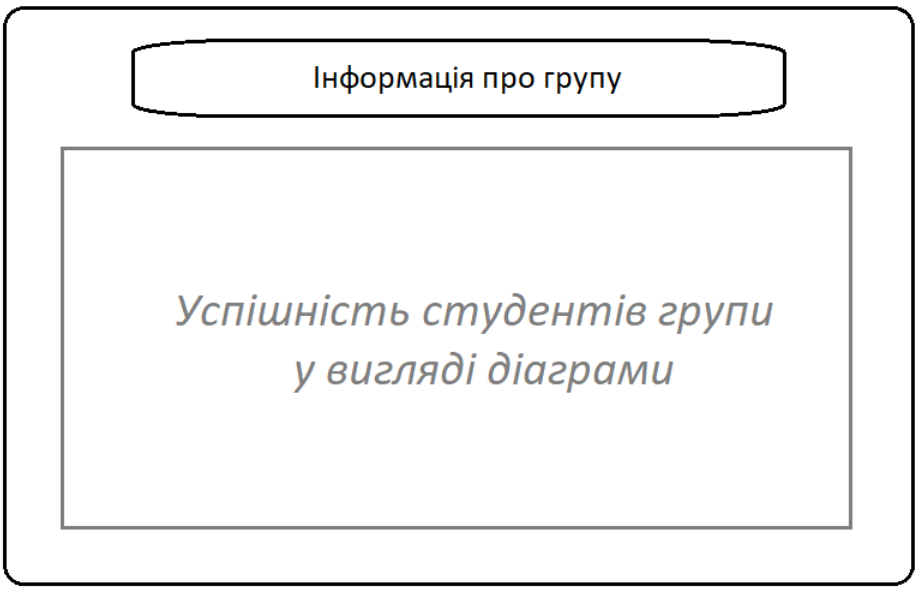


Рис. 2.11 Ескіз вікна відображення успішності студентів групи в вигляді діаграми

На рисунку 2.12 зображено діаграму граничних класів.

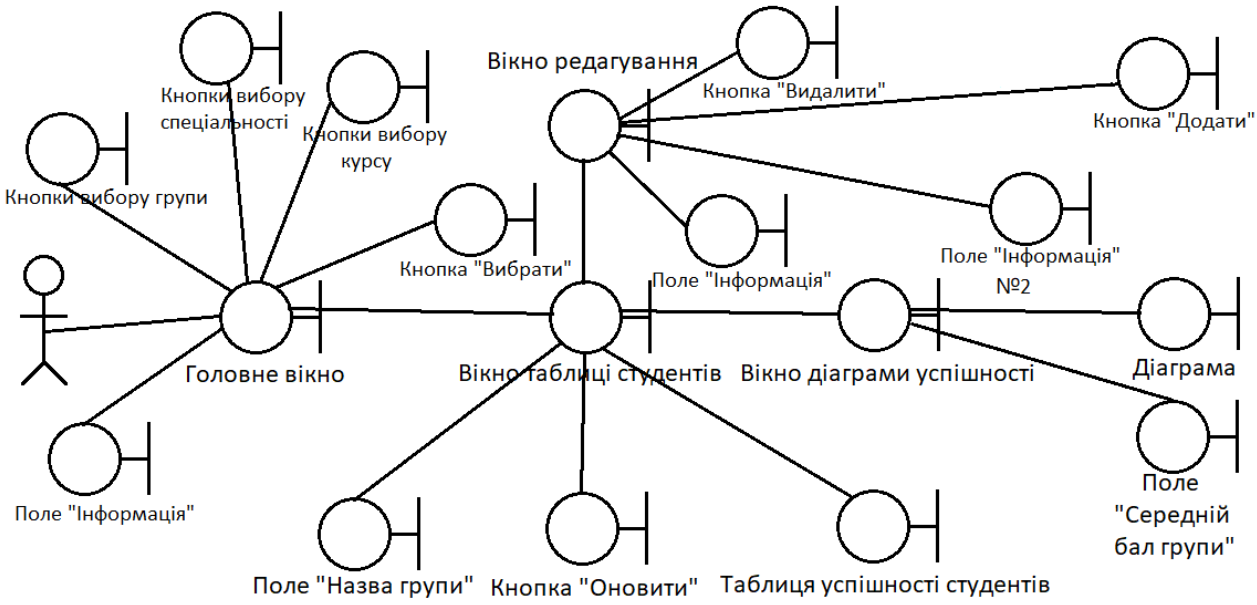


Рис. 2.12 Діаграма граничних класів

2.3 Таблиця відповідності елементів бібліотеці Tkinter

Елемент інтерфейсу	Клас, реалізуючий елемент
Головне вікно	tkinter.Tk
Поле «Інформація»	tkinter.Label
Кнопка «Вибрати»	tkinter.ttk.Button
Кнопки вибору групи	tkinter.ttk.Button
Кнопки вибору спеціальності	tkinter.ttk.Button
Кнопки вибору курсу	tkinter.ttk.Button
Вікно таблиці студентів	tkinter.Toplevel
Поле «Назва групи»	tkinter.Label
Кнопка «Оновити»	tkinter.ttk.Button
Таблиця успішності студентів	tkinter.ttk.Treeview
Вікно редагування	tkinter.Toplevel
Поле «Інформація»	tkinter.Entry
Кнопка «Додати»	tkinter.ttk.Button
Поле «Інформація» №2	tkinter.Entry
Кнопка «Видалити»	tkinter.ttk.Button
Вікно діаграми успішності	tkinter.Toplevel
Діаграма	matplotlib.figure.Figure
Поле «Середній бал групи»	tkinter.Label

3. РОЗРОБКА ПРОГРАМНОГО ДОДАТКУ

3.1. Діаграма класів

На діаграма зображено діаграму класів програми. На ній можна побачити класи, що реалізують інтерфейс користувача (View1, View2, View3, View4) та клас моделі (Model), що забезпечує коректну роботу з базою даних.

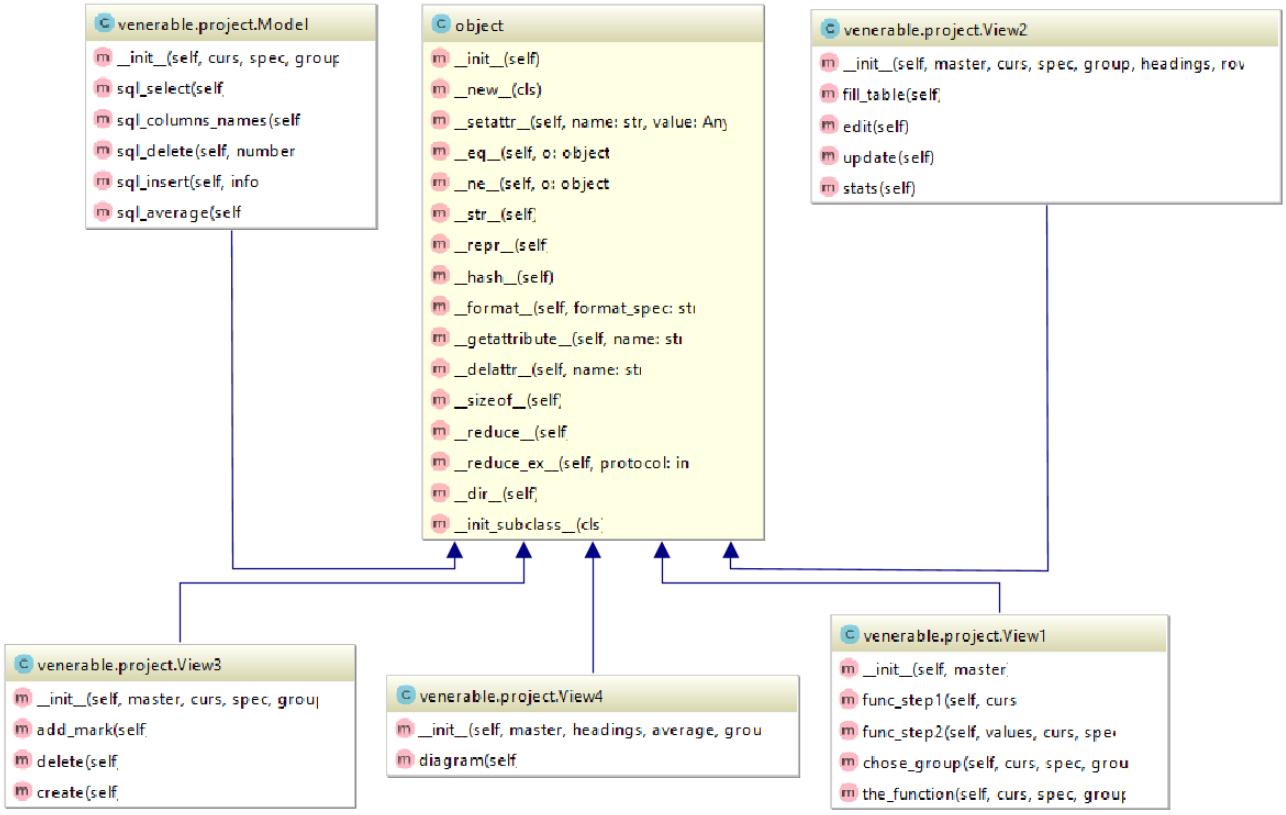


Рис. 3.1 Діаграма класів

3.2. Опис представлення

В даній курсовій роботі для візуалізації інтерфейсу користувача було використану бібліотеку Tkinter. В програмі використовується чотири вікна, для кожного з них був написаний окремий клас.

Клас View1 – клас, який реалізує візуалізацію інтерфейсу користувача в першому вікні. View1 має наступні методи:

- chose_group(self, curs, spec, group) – метод, який викликається, при натисканні на кнопку «Назва групи». Метод виводить інформацію про вибір користувача і розміщує кнопку «Вибрати».
- func_step1(self, curs) – метод, який викликається, при натисканні на одну із кнопок курсу (після вибору курсу). Метод створює і розміщує кнопки зі спеціальностями в залежності від попереднього вибору користувача.
- the_function(self, curs, spec, group) – метод, який викликається при натисканні на кнопку «Вибрати». Метод створює об’єкт моделі (Model), робить запити до бази даних. Результати запитів метод використовує як параметри при створенні нового вікна (клас View2).
- func_step2(self, values, curs, spec) – метод, який викликається, при натисканні на одну із кнопок спеціальності. Метод створює і розміщує кнопки з групами вибраної спеціальностями в залежності від попереднього вибору користувача.

4. ТЕСТУВАННЯ ПРОГРАМНОГО ЗАПЕЗПЕЧЕННЯ

Під час запуску програми, користувач бачить інтерфейс з кнопками вибору курсу, спеціальності, групи та кнопкою входу до вікна, де розташована таблиця успішності студентів учбового закладу згідно з прецедентом на Рис. 2.2. Вигляд головного вікна з кнопкою переходу до наступного вікна можна побачити на Рис. 4.1:

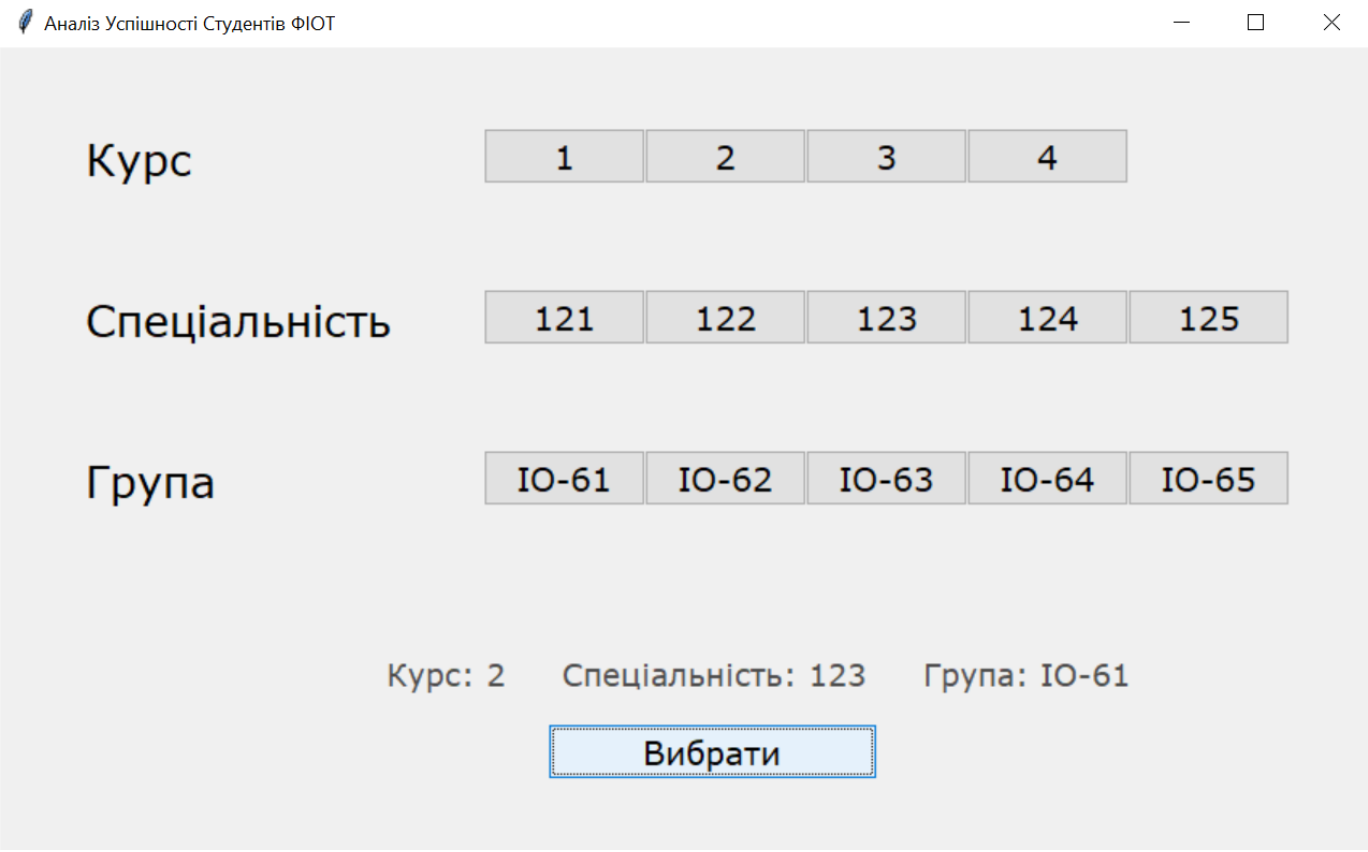
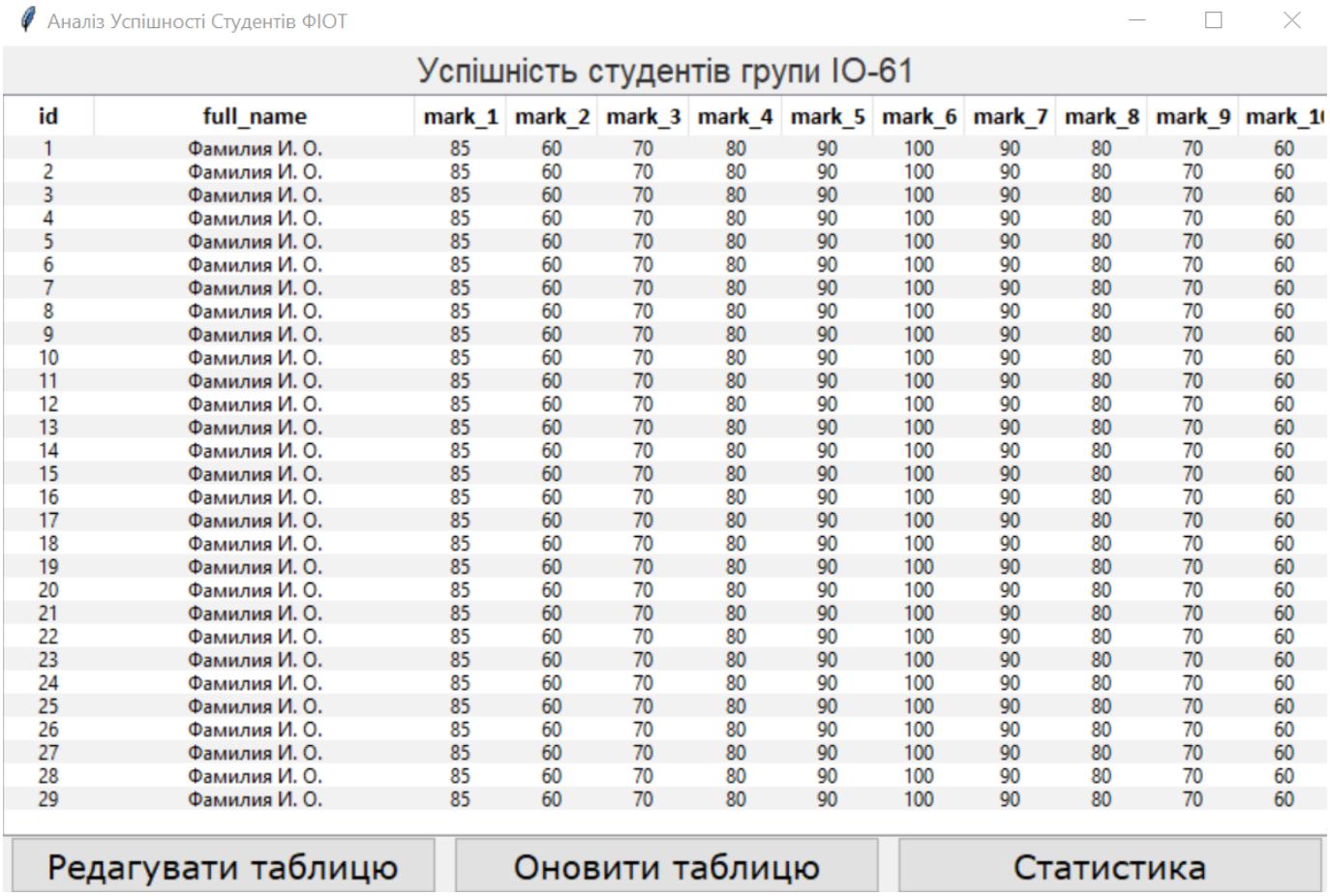


Рис. 4.1. Головне вікно програми

Користувач може обрати будь-яку функцію дій з програмним додатком, згідно з прецедентом на Рис. 2.1. Це відображає Рис. 4.2:



id	full_name	mark_1	mark_2	mark_3	mark_4	mark_5	mark_6	mark_7	mark_8	mark_9	mark_1
1	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
2	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
3	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
4	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
5	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
6	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
7	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
8	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
9	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
10	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
11	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
12	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
13	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
14	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
15	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
16	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
17	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
18	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
19	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
20	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
21	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
22	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
23	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
24	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
25	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
26	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
27	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
28	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60
29	Фамилия И. О.	85	60	70	80	90	100	90	80	70	60

Рис. 4.2. Перегляд таблиці успішності студентів та меню програми

Користувач має змогу переглянути успішність групи у вигляді діаграми, натиснувши кнопку «Статистика», згідно з прецедентом на Рис. 2.3. На Рис. 4.3 зображена операція перегляду успішності студентів в вигляді діаграми:

Рис. 4.3. Перегляд успішності студентів у вигляді діаграми

Користувач має змогу увійти в меню редагування списку студентів, натиснувши кнопку «Редагувати таблицю», згідно з прецедентом на Рис. 2.4. На Рис. 4.4. зображена вікно редагування бази даних студентів. Користувач може додати вказаного студента до групи у вікні редагування вписавши потрібну інформацію, а саме прізвище та ініціали студента, номер в списку групи та бали за запропоновані предмети. Користувач додає студента до групи кнопкою «Додати», згідно з прецедентом на Рис. 2.5. На Рис. 4.5. зображено операцію додавання студента до групи. Користувач також має можливість видалити студента з групи, вказавши потрібний номер в списку групи та натиснувши кнопку «Видалити». Операція відбувається згідно з прецедентом на Рис. 2.6.

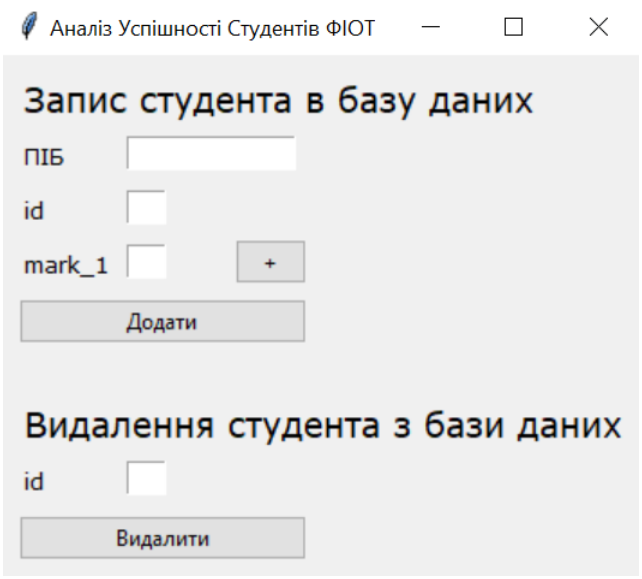


Рис. 4.4. Перегляд вікна редагування

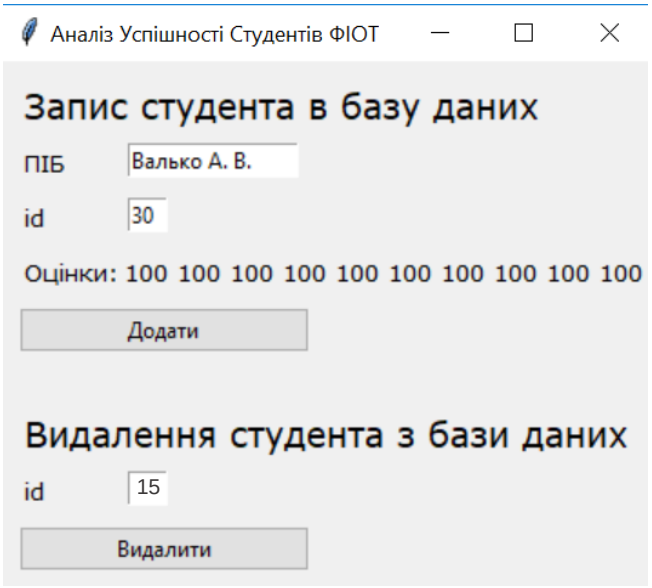


Рис. 4.5. Операція додавання та видалення студента

На Рис. 4.6 зображений результат операції додавання студента до групи. Як показано на рисунку, студенти додаються не відразу:

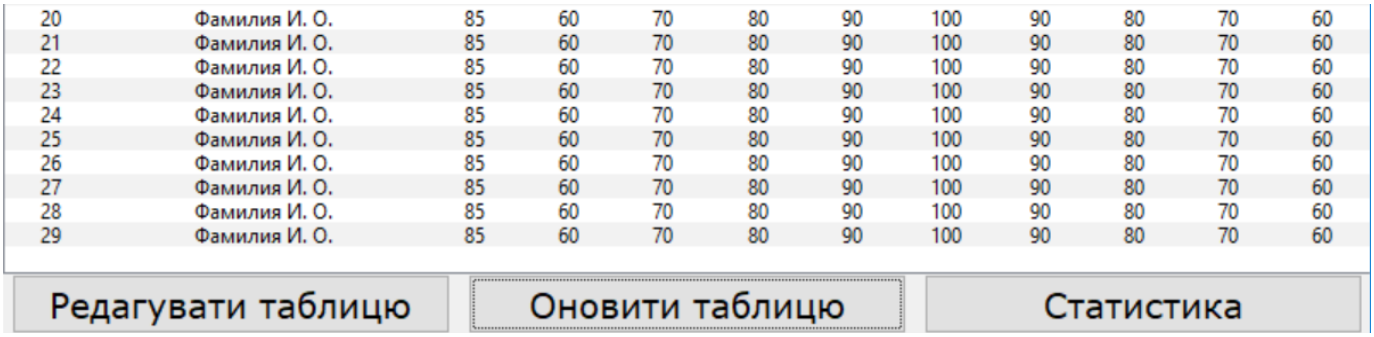


Рис. 4.6. Результат додавання студента до групи

Лише після оновлення списку студентів групи, користувач зможе побачити доданого студента. Згідно з прецедентом на Рис. 2.7 потрібно натиснути кнопку «Оновити таблицю», щоб побачити оновлений результат. На Рис. 4.7 зображено результат виконання операції оновлення списку студентів:

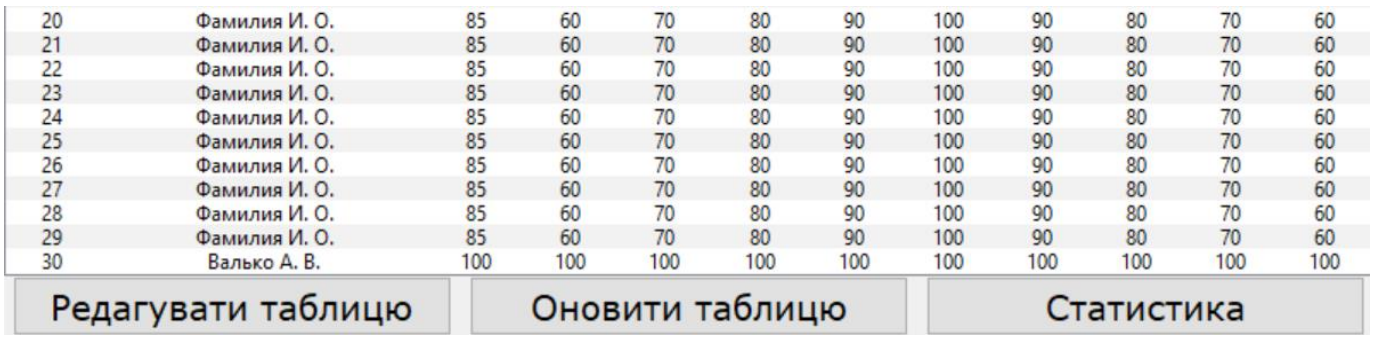


Рис. 4.7. Результат виконання операції оновлення каталогу

Операція видалення студента відбувається аналогічним чином з операцією додавання. Тобто, після видалення студента з групи, користувач не побачить змін. Потрібно натиснути кнопку «Оновити» для відображення результату, згідно з прецедентом на Рис. 2.7.

В результаті тестування програмного додатку було перевірено на працездатність а́бсолютно всі прецеденти, які були описані в частині «Проектування програмного додатку». Під час тестування всі прецеденти працюють правильно і без помилок. В даній частині курсової роботи було розроблено візуальне відображення роботи кожного прецедента.

ВИСНОВКИ

Під час виконання даної курсової роботи були закріплені знання з проектування, моделювання, тестування програмного забезпечення та були отримані навички у проектуванні та розробці програмного додатку з графічним інтерфейсом користувача, взаємодією з базами даних, було вивчено шаблон проектування MVC та основи програмування на мові Python за допомогою бібліотеки Tkinter. Розроблено графічний інтерфейс, який задовольняє вимоги проектування.

В курсовій роботі розроблена програма, яка призначена для аналізу успішності студентів учбового закладу. Основною метою було розробити додаток, який дозволяє користувачу переглядати та аналізувати успішність студентів, має функціонал для редагування списку студентів.

В роботі розглядаються принципи побудови графічного інтерфейсу користувача, його реалізація на мові Python в середовищі PyCharm. Робота містить повну документацію для коду програми.

ЛІТЕРАТУРА

1. Марк Лутц Программирование на Python. Том 1, 4-е издание – «Символ-Плюс», 2011 – 992 с. — ISBN 978-1-449-35573-9.

2. Бандура, В. В. Архітектура та проектування програмного забезпечення конспект лекцій / В. В. Бандура, Р. І. Храбатин. - Івано-Франківськ: ІФНТУНГ, 2012. - 240 с.

3. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object-Oriented Software. — СПб: «Питер», 2007. — С. 366. — ISBN 978-5- 469-01136- 1 (также ISBN 5-272-00355-1).

4. http://uk.wikipedia.org/wiki/Шаблони проектування програмного забезпечення.

ДОДАТКИ

ДОДАТОК 1: ПРОГРАМНИЙ КОД ПРОЕКТУ

```
from tkinter import Label, Toplevel, CENTER, END, Entry, Tk
from tkinter.ttk import Treeview, Style, Button as Button_ttk
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from numpy import arange
import sqlite3


class Model:
    """ Взаимодействие с базами данных """
    def __init__(self, curs, spec, group):
        self.curs = curs
        self.spec = spec
        self.group = group

    def sql_select(self):
        """Есть папки курсов. В них есть базы данных специальностей этого курса. В
каждой бд есть таблицы групп
        Например: curs_2/spec_123.db (Таблица io61 в этой базе данных)"""
        conn = sqlite3.connect('database\\curs_{}\\spec{}_v2.db'.format(self.curs,
self.spec))  # конект к базе данных
        cursor = conn.execute('SELECT * FROM {} ORDER BY
id;'.format(self.group.replace('-', '').lower()))  # запрос
        result = cursor.fetchall()  # в переменную присваиваем результата запроса
        # print(result)
        conn.close()  # дисконектимся от базы данных

        return result

    def sql_columns_names(self):
        """Возвращает названия столбцов"""
        conn = sqlite3.connect('database\\Curs_{}\\spec{}_v2.db'.format(self.curs,
self.spec))
        cursor = conn.execute('SELECT * FROM {};'.format(self.group.replace('-',
'').lower()))  # делаем запрос
        columns_names = [desc[0] for desc in cursor.description]  # получаем название
стобцов
        # print(columns_names)
        conn.close()  # дисконектимся от базы данных

        return columns_names

    def sql_delete(self, number):
        """ Удаление студента с базы данных """
        conn = sqlite3.connect('database\\curs_{}\\spec{}_v2.db'.format(self.curs,
self.spec))
        conn.execute('DELETE FROM {} WHERE id = {};'.format(self.group.replace('-',
'').lower(), number))
        conn.commit()
        # print('deleted')
        conn.close()
```

```
    def sql_insert(self, info):
        """ Добавление студента в базу данных """
        conn = sqlite3.connect('database\\curs_{}\\spec{}_v2.db'.format(self.curs,
self.spec))
        # print('INSERT INTO {} VALUES (?, ?, ?);'.format(group.replace('-',
'').lower(), number, name, marks))
        conn.execute('INSERT INTO {} VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);'.
                     format(self.group.replace('-', '').lower()), info)
        conn.commit()
        # print('inserted')
        conn.close()

    def sql_average(self):
        conn = sqlite3.connect('database\\Curs_{}\\spec{}_v2.db'.format(self.curs,
self.spec))  # конект к базе данных
        cursor = conn.execute('SELECT AVG(mark_1), AVG(mark_2), AVG(mark_3),
AVG(mark_4), AVG(mark_5), AVG(mark_6),'
                              ' AVG(mark_7), AVG(mark_8), AVG(mark_9), AVG(mark_10)
FROM {};'
                              .format(self.group.replace('-', '').lower()))  # делаем
запрос
        result = cursor.fetchall()
        # print(result)
        conn.close()  # дисконектимся от базы данных
        return result


class View1:
    """ Главное окно """
    options_2 = ('121', '122', '123', '124', '125')  # специальности
    # все группы каждой специальности
    options_121 = (('ИП-71', 'ИП-72', 'ИП-73', 'ИП-74', 'ИП-75'), ('ИП-61', 'ИП-62',
'ИП-63', 'ИП-64', 'ИП-65'),
                   ('ИП-51', 'ИП-52', 'ИП-53', 'ИП-54', 'ИП-55'), ('ИП-41', 'ИП-42',
'ИП-43', 'ИП-44', 'ИП-45'))

    options_122 = (('ИС-71', 'ИС-72', 'ИС-73', 'ИС-74', 'ИС-75'), ('ИС-61', 'ИС-62',
'ИС-63', 'ИС-64', 'ИС-65'),
                   ('ИС-51', 'ИС-52', 'ИС-53', 'ИС-54', 'ИС-55'), ('ИС-41', 'ИС-42',
'ИС-43', 'ИС-44', 'ИС-45'))

    options_123 = (('ИО-71', 'ИО-72', 'ИО-73', 'ИО-74', 'ИО-75'), ('ИО-61', 'ИО-62',
'ИО-63', 'ИО-64', 'ИО-65'),
                   ('ИО-51', 'ИО-52', 'ИО-53', 'ИО-54', 'ИО-55'), ('ИО-41', 'ИО-42',
'ИО-43', 'ИО-44', 'ИО-45'))

    options_124 = (('ИК-71', 'ИК-72', 'ИК-73', 'ИК-74', 'ИК-75'), ('ИК-61', 'ИК-62',
'ИК-63', 'ИК-64', 'ИК-65'),
                   ('ИК-51', 'ИК-52', 'ИК-53', 'ИК-54', 'ИК-55'), ('ИК-41', 'ИК-42',
'ИК-43', 'ИК-44', 'ИК-45'))

    options_125 = (('ИТ-71', 'ИТ-72', 'ИТ-73', 'ИТ-74', 'ИТ-75'), ('ИТ-61', 'ИТ-62',
'ИТ-63', 'ИТ-64', 'ИТ-65'),
                   ('ИТ-51', 'ИТ-52', 'ИТ-53', 'ИТ-54', 'ИТ-55'), ('ИТ-41', 'ИТ-42',
'ИТ-43', 'ИТ-44', 'ИТ-45'))

    def __init__(self, master):
```

```
        # Настраиваем характеристика главного окна
        self.master = master
        # master.overrideredirect(1)
        # master.geometry("%dx%d+0+0" % (root.winfo_screenwidth(),
root.winfo_screenheight()))
        # master['bg'] = 'grey90'
        master.title('Аналіз Успішності Студентів ФІОТ')
        master.wm_geometry("%dx%d+%d+%d" % (850, 500, 225, 70))  # размер окна +
расположение
        fon = 'Verdana 20'
        fon2 = 'Verdana 14'
        s = Style()
        s.configure('my.TButton', font='Verdana 15')

        # надписи
        self.label_curs = Label(master, text="Курс", font=fon).place(x=50, y=50)
        self.label_spec = Label(self.master, text="Спеціальність ",
font=fon).place(x=50, y=150)
        self.label_group = Label(master, text="Група", font=fon).place(x=50, y=250)
        # кнопочки (выбираем курс)
        # style = Style()
        # style.configure('BW.TLabel', foreground='black', background='white',
font='Arial 20')
        self.my_button1 = Button_ttk(master, text='1', command=lambda:
self.func_step1(1), style='my.TButton', width=7)
        self.my_button2 = Button_ttk(master, text='2', command=lambda:
self.func_step1(2), style='my.TButton', width=7)
        self.my_button3 = Button_ttk(master, text='3', command=lambda:
self.func_step1(3), style='my.TButton', width=7)
        self.my_button4 = Button_ttk(master, text='4', command=lambda:
self.func_step1(4), style='my.TButton', width=7)
        self.my_button5 = Button_ttk(master)
        self.my_button1.place(x=300, y=50)
        self.my_button2.place(x=400, y=50)
        self.my_button3.place(x=500, y=50)
        self.my_button4.place(x=600, y=50)

        # лейбл, который отображает, что вы выбрали в первом окне
        self.label_result = Label(self.master, font=fon2, foreground='gray30')
        self.label_result.place(x=237, y=375)

        self.go_button = None  # кнопка 'вибрати'
        self.new_window = None  # второе окно делаем
        self.app = None  # второе окно

    # Функции первого шага
    def func_step1(self, curs):
        """ Функция, которая вызывается при выборе курса
        Создает и размещает кнопки с нужными специальностями """
        self.my_button1 = Button_ttk(self.master, text=self.options_2[0],
style='my.TButton', width=7,
                                     command=lambda:
self.func_step2(self.options_121[curs-1], curs, self.options_2[0]))
        self.my_button2 = Button_ttk(self.master, text=self.options_2[1],
style='my.TButton', width=7,
                                     command=lambda:
self.func_step2(self.options_122[curs-1], curs, self.options_2[1]))
        self.my_button3 = Button_ttk(self.master, text=self.options_2[2],
```

```
style='my.TButton', width=7,
                                    command=lambda:
self.func_step2(self.options_123[curs-1], curs, self.options_2[2]))
        self.my_button4 = Button_ttk(self.master, text=self.options_2[3],
style='my.TButton', width=7,
                                    command=lambda:
self.func_step2(self.options_124[curs-1], curs, self.options_2[3]))
        self.my_button5 = Button_ttk(self.master, text=self.options_2[4],
style='my.TButton', width=7,
                                    command=lambda:
self.func_step2(self.options_125[curs-1], curs, self.options_2[4]))

        self.my_button1.place(x=300, y=150)
        self.my_button2.place(x=400, y=150)
        self.my_button3.place(x=500, y=150)
        self.my_button4.place(x=600, y=150)
        self.my_button5.place(x=700, y=150)

    # Функции второго шага
    def func_step2(self, values, curs, spec):  # параметр values - это список/котреж
групп. ex: ('IO-61', 'IO-62'..)
        """ Функция, которая вызывается, при выборе специальности
        Создает и размещает кнопки с группами выбраной специальности"""
        self.my_button1 = Button_ttk(self.master, text=values[0], style='my.TButton',
width=7,
                                    command=lambda: self.chose_group(curs, spec,
values[0]))
        self.my_button2 = Button_ttk(self.master, text=values[1], style='my.TButton',
width=7,
                                    command=lambda: self.chose_group(curs, spec,
values[1]))
        self.my_button3 = Button_ttk(self.master, text=values[2], style='my.TButton',
width=7,
                                    command=lambda: self.chose_group(curs, spec,
values[2]))
        self.my_button4 = Button_ttk(self.master, text=values[3], style='my.TButton',
width=7,
                                    command=lambda: self.chose_group(curs, spec,
values[3]))
        self.my_button5 = Button_ttk(self.master, text=values[4], style='my.TButton',
width=7,
                                    command=lambda: self.chose_group(curs, spec,
values[4]))
        self.my_button1.place(x=300, y=250)
        self.my_button2.place(x=400, y=250)
        self.my_button3.place(x=500, y=250)
        self.my_button4.place(x=600, y=250)
        self.my_button5.place(x=700, y=250)

    # Функции третьего шага
    def chose_group(self, curs, spec, group):
        """Функция, которая вызывается при нажатии на <> группу"""
        self.label_result['text'] = 'Курс: {}     Спеціальність: {}     Група:
{}'.format(curs, spec, group)
        self.go_button = Button_ttk(self.master, text='Вибрати', style='my.TButton',
width=15,
                                    command=lambda: self.the_function(curs, spec,
group))
```

```
        self.go_button.place(x=340, y=420)
        # print('Ви вибрали: {} курс {} спеціальність {} група'.format(curs, spec,
group))

    def the_function(self, curs, spec, group):
        """Функция, которая вызывается при нажатии на <Вибрати> """
        simple = Model(curs, spec, group)
        students = simple.sql_select()  # students - список кортежей. ex: [('1',
'2'), ('3', '4')..]
        columns = simple.sql_columns_names()  # columns - список названий столбцов

        # Создаем новое окно
        self.new_window = Toplevel(self.master)
        self.app = View2(self.new_window, curs, spec, group, columns, students)


class View2:
    """ Окно, которое открывается при нажатии на 'Вибрати' """
    def __init__(self, master, curs, spec, group, headings, rows):
        self.master = master
        self.curs = curs
        self.spec = spec
        self.group = group
        self.headings = headings
        self.rows = rows
        self.average = None
        master.wm_geometry("%dx%d+%d+%d" % (800, 511, 250, 70))

        style = Style()
        style.configure("mystyle.Treeview", highlightthickness=1, bd=1, rowheight=14)
        style.configure("mystyle.Treeview.Heading", font=('Calibri', 11, 'bold'))  #
Modify the font of the headings

        self.my_table = Treeview(master, show='headings', height=30,
style="mystyle.Treeview")  # создаем таблицу
        self.fill_table()  # заполняем таблицу
        self.my_table.grid(row=1, column=0, columnspan=3)  # выводим таблицу

        self.label_info = Label(master, text='Успішність студентів групи
{}'.format(group), font='Arial 15',
                                foreground='gray20')
        self.label_info.grid(row=0, column=0, columnspan=3)

        self.eButton = Button_ttk(master, text='Редагувати таблицю', width=19,
command=self.edit, style='my.TButton')
        self.eButton.grid(row=2, column=0)

        self.updButton = Button_ttk(master, text='Оновити таблицю', width=19,
command=self.update, style='my.TButton')
        self.updButton.grid(row=2, column=1)

        self.button_stat = Button_ttk(master, text='Статистика', width=19,
command=self.stats, style='my.TButton')
        self.button_stat.grid(row=2, column=2)

        self.new_window = None
        self.app = None
```

```
    def fill_table(self):
        """ Работаем с таблицой """
        self.my_table['columns'] = self.headings  # в 'столбцы' присваиваем
переменную

        for head in self.headings:
            """ Создаем столбцы. Если это стоблец П.И.Б. - делаем ширину больше """
            if head == 'full_name':
                self.my_table.heading(head, text=head, anchor=CENTER)
                self.my_table.column(head, anchor=CENTER, width=193)
            else:
                self.my_table.heading(head, text=head, anchor=CENTER)
                self.my_table.column(head, anchor=CENTER, width=55)

        counter = 0
        self.my_table.tag_configure('odd', background='#FFFFFF')
        self.my_table.tag_configure('even', background='#F2F2F2')
        for row in self.rows:
            if counter % 2 == 1:
                self.my_table.insert('', END, values=row, tags=('odd',))  # вставляем
в каждую строку таблицы
            else:
                self.my_table.insert('', END, values=row, tags=('even',))
            counter += 1

    def edit(self):
        """ Редактировать таблицу """
        self.new_window = Toplevel(self.master)
        self.app = View3(self.new_window, self.curs, self.spec, self.group)

    def update(self):
        """ Обновить таблицу (Заново берем данные с базы данных и создаем новую
таблицу) """
        simple = Model(self.curs, self.spec, self.group)
        self.rows = simple.sql_select()
        self.headings = simple.sql_columns_names()

        self.my_table = Treeview(self.master, show='headings', height=30,
style="mystyle.Treeview")
        self.fill_table()  # заполняем таблицу
        self.my_table.grid(row=1, column=0, columnspan=3)  # выводим таблицу

    def stats(self):
        """ Новое окно, где размещается диаграмма """
        simple = Model(self.curs, self.spec, self.group)
        self.headings = simple.sql_columns_names()[2::]
        self.average = simple.sql_average()[0]
        self.new_window = Toplevel(self.master)
        self.app = View4(self.new_window, self.headings, self.average, self.group)


class View3:
    fon_title = 'Verdana 14'
    fon_normal = 'Verdana 10'

    def __init__(self, master, curs, spec, group):
        self.master = master
        self.curs = curs
```

```
        self.spec = spec
        self.group = group
        self.marks = []
        self.counter = 2
        self.simple = Model(self.curs, self.spec, self.group)
        self.headings = self.simple.sql_columns_names()
        master.wm_geometry("%dx%d+%d+%d" % (355, 290, 800, 200))

        # Функционал для добавления значения (labels, entries, buttons)
        self.label_caption = Label(master, text='Запис студента в базу даних',
font=self.fon_title)
        self.label_caption.place(x=10, y=10)

        self.label_create_1 = Label(master, text='ПІБ', font=self.fon_normal)
        self.label_create_1.place(x=10, y=45)
        self.entry_create_1 = Entry(master, width=15)
        self.entry_create_1.place(x=70, y=45)

        self.label_create_2 = Label(master, text='id', font=self.fon_normal)
        self.label_create_2.place(x=10, y=75)
        self.entry_create_2 = Entry(master, width=3)
        self.entry_create_2.place(x=70, y=75)

        self.label_create_3 = Label(master, text=self.headings[self.counter],
font=self.fon_normal)
        self.label_create_3.place(x=10, y=105)
        self.entry_create_3 = Entry(master, width=3)
        self.entry_create_3.place(x=70, y=105)

        self.button_create_add = Button_ttk(master, text='+', command=self.add_mark,
width=5)
        self.button_create_add.place(x=130, y=102)

        self.button_create_final = Button_ttk(master, text='Додати', width=25,
command=self.create)
        self.button_create_final.place(x=10, y=135)

        self.label_create_0 = Label(master, text='Оцінки:', font=self.fon_normal)
        self.label_info_1 = Label(master, text='Помилка!', foreground='gray30',
font=self.fon_normal)

        # Функционал для удаления значения (labels, entries, buttons)
        self.label_caption_delete = Label(master, text='Видалення студента з бази
даних', font=self.fon_title)
        self.label_caption_delete.place(x=10, y=190)

        self.label_delete = Label(master, text='id', font=self.fon_normal)
        self.label_delete.place(x=10, y=225)

        self.entry_delete = Entry(master, width=3)
        self.entry_delete.place(x=70, y=225)

        self.button_delete = Button_ttk(master, text='Видалити', width=25,
command=self.delete)
        self.button_delete.place(x=10, y=255)

        self.label_info_2 = Label(master, text='Помилка!', foreground='gray30',
font=self.fon_normal)
```

```
    def add_mark(self):
        self.counter += 1
        self.marks.append(self.entry_create_3.get())
        # print(self.marks)
        self.label_create_3.destroy()
        self.entry_create_3.destroy()
        try:
            self.label_create_3 = Label(self.master,
text=self.headings[self.counter], font=self.fon_normal)
        except IndexError:
            self.label_create_3.destroy()
            self.entry_create_3.destroy()
            self.button_create_add.destroy()
            self.label_create_0.place(x=10, y=105)
            self.label_create_3 = Label(self.master, text=self.marks,
font=self.fon_normal)
            self.label_create_3.place(x=65, y=105)
        else:
            self.label_create_3.place(x=10, y=105)
            self.entry_create_3 = Entry(self.master, width=3)
            self.entry_create_3.place(x=70, y=105)

    def delete(self):
        # print('deleting')
        try:
            student_id = self.entry_delete.get()  # получаем id, которое ввел
пользователь
            self.simple.sql_delete(student_id)  # вызываем функцию, которая удаляет
студента
        except sqlite3.OperationalError:
            self.label_info_2.place(x=200, y=255)
        else:
            self.label_info_2.destroy()

    def create(self):
        try:
            name = self.entry_create_1.get()
            number = self.entry_create_2.get()
            info = self.marks  # строка. пример: '90 95 65 70 87 100..'

            info.insert(0, name)  # в начало списка вставляем ФИО студента
            info.insert(0, number)  # также в начало вставляем номер студента
            # print(info)  # переменная info выглядит след. образом: [id, 'Full
Name', 90, 65, 70 ...]

            self.simple.sql_insert(info)
        except sqlite3.ProgrammingError:
            self.label_info_1.place(x=200, y=135)
        else:
            self.label_info_1.destroy()


class View4:
    """ Окно, где размещается диаграмма """
    def __init__(self, master, headings, average, group):
        self.master = master
        self.headings = headings
```

```
        self.average = average
        self.group = group
        master.wm_geometry("%dx%d+%d+%d" % (600, 450, 450, 100))

        self.fig = self.diagram()
        canvas = FigureCanvasTkAgg(self.fig, master)
        canvas.show()
        canvas.get_tk_widget().pack()

    def diagram(self):
        """ Построение гистрограммы """
        self.headings = sorted(self.headings)  # для коректного отображения чисел на гистограмме
        # print(self.headings, '\n', self.average)
        num = arange(1, len(self.headings) + 1)  # список значений оси x
        average_group_mark = round(sum(self.average)/len(self.average), 2)  # Средний балл по группе по всем предметам
        figure = Figure()
        # название графика (сверху)
        figure.suptitle('Успішність студентів групи {}\nСередній бал: {}'.format(self.group, average_group_mark))
        ax = figure.add_subplot(111)  # создаем полотно
        ax.bar(self.headings, self.average)  # гистограмма с значениями x и y
        figure.autofmt_xdate(bottom=0.2, rotation=50)  # формат подписей снизу

        for x, y in zip(num, self.average):
            ax.text(x - 1.35, y - 5, '%.1f' % y, color='white')  # подписываем каждый столбец

        return figure


if __name__ == '__main__':
    root = Tk()
    my_gui = View1(root)
    root.mainloop()
```