Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Цуркан Ганна Сергіївна,
студентка групи АС-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Програмний засіб для візуалізації властивостей освітньої програми

Спеціальність:
121 – Інженерія програмного забезпечення

Освітня програма:
Інженерія програмного забезпечення

Керівник:
Любченко Віра Вікторівна,
докт. техн. наук, професор

Одеса – 2021

ЗМІСТ

Міністерство освіти і науки України
Одеський національний політехнічний університет
Навчально-науковий інститут комп'ютерних систем
Кафедра системного програмного забезпечення

Рівень вищої освіти: другий (магістерський)
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ Любченко В. В.
«_ __»_____ 20__р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Цуркан Ганни Сергіївни, студентки групи АС-161

1. Тема роботи: Програмний засіб для візуалізації властивостей освітньої програми

Керівник роботи: Любченко Віра Вікторівна, докт. техн. наук, професор

затверджені наказом ректора від 25 жовтня 2021 р. № 374-в

2. Зміст роботи:
Вступ, аналіз проблеми, методика проведення досліджень, технічне завдання програмного засобу, проектування програмного засобу, програмна реалізація програмного засобу, тестування програмного засобу, висновки.

3. Перелік ілюстративного матеріалу
Відповідно до слайдів презентації: мета роботи, аналоги програмного засобу, архітектура програмного засобу, діаграма програмних класів, алгоритм отримання даних для діаграми з бази даних, інтерфейс користувача, ефективність програмного засобу, висновки

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

4. Дата видачі завдання: «01» вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	Аналіз предметної області	15.09.2021	Вик.
2	Специфікація вимог до програмного засобу	01.10.2021	Вик.
3	Проєктування програмного засобу	15.10.2021	Вик.
4	Реалізація програмного засобу	15.11.2021	Вик.
5	Тестування програмного засобу	01.12.2021	Вик.

Здобувач вищої освіти _____ Г. С. Цуркан

Керівник роботи _____ В. В. Любченко

АНОТАЦІЯ

Метою роботи є зменшення кількості годин аналізу властивостей освітньої програми, який відобразить залежність між дисциплінами та результатами навчання, за певний період часу, за допомогою візуалізації даних у вигляді діаграм. Технологіями розробки є мова програмування Python, програмне розширення фреймворку графічних інтерфейсів Qt для мови програмування Python – PyQt, реляційна база даних PostgreSQL. Як результат роботи створений програмний засіб для візуалізації властивостей освітньої програми у вигляді різноманітних діаграм – Visu, який допомагає виявити залежність між певними дисциплінами та їх результатами навчання.

Ключові слова: аналіз освітньої програми, візуалізація, діаграми, Python, PyQt, PostgreSQL.

ABSTRACT

The purpose of the work is to reduce the number of hours for educational program properties analysis that will reflect the relationship between disciplines and their learning outcomes over a period of time with diagram data visualization. The development technologies are a programming language Python, a software extension of the Qt GUI framework for the Python programming language – PyQt, PostgreSQL relational database. The result of the work is a software tool for an educational program properties visualization in various diagrams – Visu, which helps identify the relationship between specific disciplines and their learning outcomes.

Keywords: educational program analysis, visualization, diagrams, Python, PyQt, PostgreSQL.

ВСТУП

Основою будь-якого навчання у закладах освіти, таких як школа, інститути, університети тощо, є освітня програма – комплекс освітніх компонентів, які сплановані та організовані для досягнення визначених результатів навчання [1]. В ролі освітніх компонентів (ОК) виступають будь-які дисципліни, курсові проєкти, практики та атестація, а результатами навчання (РН) вважають сукупність знань, умінь, навичок, що були набуті під час навчання і можуть бути кількісно оцінені та вимірювані. Існують різні види освітніх програм, і до кожного встановлюються конкретні вимоги. Після складання програм, вони схвалюються Вченою радою закладу освіти і затверджуються керівником Ради. Проте не завжди можна скласти освітню програму, яка в повній мірі буде задовольняти потреби здобувачів освіти. Після затвердження освітньої програми з часом випливають певні проблеми, такі як незбалансованість дисциплін в розрізі загальної освітньої програми, або нестача конкретних дисциплін для освітньої програми для вузькоспеціалізованих напрямків, або неправильний порядок викладання дисциплін, що в цілому знижують якість навчання для здобувачів освіти. Можуть бути випадки, коли для одного предмета вивчення формуються багато результатів навчання, а для іншого – мало, і недостатня кількість результатів навчання призводить до неповної оцінки здобутих знань навчаючих.

Робота присвячена розробці програмного забезпечення Visu, яке візуально відобразить залежність між компонентами освітньої програми і відповідних результатів навчання у вигляді графіків, що допоможуть викладачам побачити недоліки складеної освітньої програми, такі як незбалансованість між дисциплінами та результатами навчання, збільшення чи зменшення кількості певних дисциплін між різними навчальними роками.

Актуальність роботи в тому, що викладачі при створенні освітньої програми за допомогою програмного засобу зможуть побачити представлення складеної освітньої

програми або попередніх освітніх програм і вирахувати певну статистику для дисциплін та результатів, що допоможе в подальшому збільшити якість освіти.

Мета роботи – це зменшення кількості годин аналізу властивостей освітньої програми, які відображають залежність між дисциплінами та результатами навчання за певний період часу, за допомогою візуалізації даних у вигляді діаграм.

Для виконання даної мети слід вирішити такі завдання:

а) представлення діаграм залежності між дисциплінами та результатами навчання у вигляді діаграм розподілу частот, bar-діаграм, spider-діаграм;

б) завантаження даних у вигляді .csv файлу.

Практична цінність одержаних результатів полягає в тому, що зараз немає програмних систем чи засобів, які були б націлені саме на візуалізацію властивостей освітньої програми, а саме відношенні освітніх компонентів до їх результатів навчання на протязі певного періоду часу (семестрів). У звичайних програмних системах для візуалізації даних немає можливості розрахувати скільки, наприклад, є результатів навчання для однієї дисципліни у навчальному році і відобразити результат на діаграмі.

Робота складається з шести розділів. В першому розділі описується предметна область та виявлені проблеми, що в ній існують. Окрім цього, в першому розділі проведений аналіз існуючих програмних рішень, які мають як свої переваги, так і недоліки, перед програмних засобом Visu. У другому розділі описана методика проведення дослідження кваліфікаційної роботи. У наступному розділі описані вимоги до програмного засобу: функціональні, нефункціональні, апаратні та програмні. Четвертий розділ присвячений проектуванню програмної системи, де вибрана архітектура та технології, які будуть використовуватися у програмній реалізації Visu. У п’ятому розділі наведені алгоритми програмного засобу, описана організація даних та відображено як виглядає інтерфейс користувача. Останній розділ кваліфікаційної роботи присвячений тестуванню програмного засобу.

1 АНАЛІЗ ПРОБЛЕМИ

Не завжди під час складання або схвалення освітньої програми, можна передбачити усі її недоліки. Наприклад, незбалансована структура загальної освітньої програми, коли навчальних предметів технічного характеру більше ніж гуманітарного, призводить до того, що у здобувачів освіти виникає недостача знань для формування грамотності. Є інші випадки, коли в освітній програмі недостатньо профільних дисциплін, і таким чином здобувачі освіти не отримують знання в передбаченій кількості та якості у зазначеній профільній області. Окрім незбалансованої структури освітньої програми, існує також проблема в некоректному порядку викладання певних дисциплін, коли дисципліна А вимагає певних знань с дисципліни Б, а дисципліни Б ще не викладалася, і таким чином виникають труднощі для здобувачів освіти в освоєнні певних тем. Також існує проблема, коли для дисципліни визначена мала кількість результатів навчання, що призводить до неясності, чи отримали здобувачі освіти достатньо знань та навичок для зазначеної дисципліни.

Для проведення детального аналізу освітньої програми, найкращим та найшвидшим рішенням є візуалізація властивостей програми, тобто відображення залежностей між дисциплінами та результатами навчання. Візуалізація представляє собою графічне представлення певної інформації, яка сприймається набагато швидше, ніж у текстовому форматі [2]. Способів візуалізації багато: графіки, діаграми, інфографіка, схеми, карти, чарти, павуки тощо, і програмних рішень для візуалізації у наш час також велика кількість. Важливо пам’ятати, що успіх візуалізації залежить, в першу чергу, від правильного вибору способу візуалізації, а вже потім – від його оформлення.

Програмні рішення для візуалізації даних можуть бути різних видів, як комп’ютерні програми, які треба встановлювати на своєму комп’ютері, так і онлайн-застосунки, які зберігаюсь усі дані на хмарних серверах. Проте такі програми

використовують одні і ті ж техніки для своєї роботи: лінійні графіки, стовпчасті графіки, гістограми, графіки розсіювання, колові, бульбашкові тощо [3].

В даній роботі було відібрано три найбільш розповсюджених аналоги розроблюваного програмного засобу. Порівняння аналогів та програмного засобу відображено у табл. 1.1 за найзначущими для вибраної предметної області, критеріями:

а) безкоштовність;

б) файлова загрузка даних – можливість не вводити дані вручну, а загрузити їх з файлу з відповідним розширенням .xls, .csv, .json тощо;

в) розрахунки – можливість проводити певні розрахунки над даними і виводити результат розрахунків на графіках;

г) збереження результатів – можливість зберегти готовий графічний результат в окремий файл формату .png, .jpg, .jpeg тощо;

д) кастомізація – можливість змінювати самостійно певні параметри діаграми чи графіку (колір, розмір тощо).

Таблиця 1.1 – Аналіз аналогів програмної системи Visu

Назва	Безкоштов-ність	Файлова загрузка даних	Розрахунки	Збереження результатів	Кастомізація
ChartBlocks	-	+	-	+	+
Microsoft Excel	+	-	-	+	+
RawGraphs	-	+	-	-	+
Visu	+	+	+	-	-

Розглянемо перший аналог ChartBlocks детальніше. ChartBlocks – онлайн програма для візуалізації даних у вигляді діаграм і графіків, ціна використання якої 20$ в місяць. Програма дає можливість користувачу як вручну вносити дані у таблиці, так і загрузити готовий набір даних з файлів з .csv, .xls розширенням. Після введення даних користувачу пропонується вибрати необхідний тип діаграми, для якої можна змінювати параметри – колір, розмір, назви стовпців, додати текст. Усі види діаграм, які доступні в програмі ChartBlocks зображені на рис.1.1:

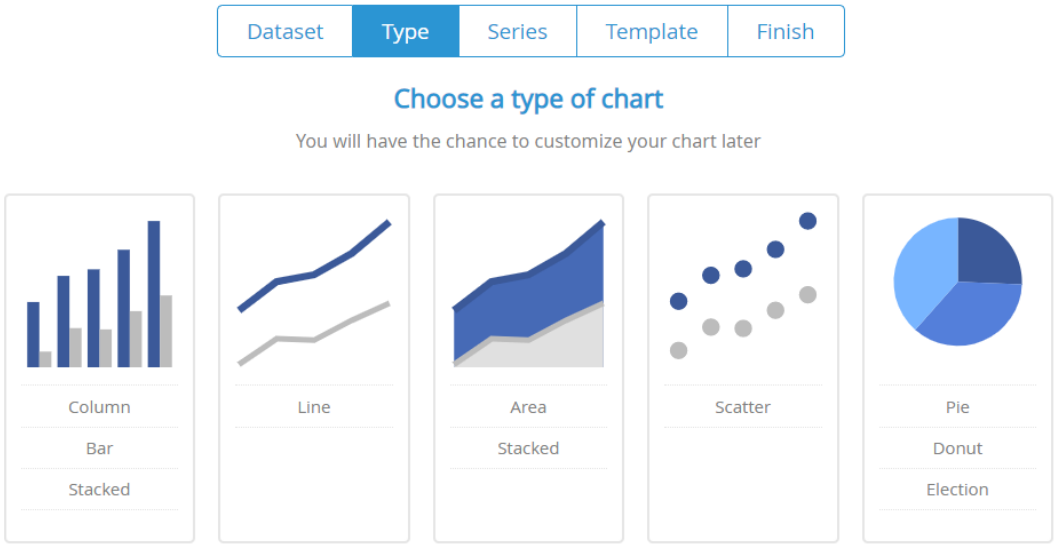

Рисунок 1.1 – Види діаграм, доступні у програмі ChartBlocks

Як бачимо з рис.1.1 користувачу доступні стовпчасті діаграми – гістограми (Column), горизонтальні діаграми (Bar), складені гістограми (Stacked). Складені гістограми означають, що на одному стовпчику зображені результати у відсотках із загального обсягу даних. Також в ChartBlocks можна відображати дані у вигляді лінійних графіків (Line), діаграм розсіювання (Scatter), кругової діаграми (Pie), кругової діаграми з розташуванням даних у відсотках (Donut), діаграми області, яка заснована на лінійному графіку і область даних між осями та лінією заштрихована або має власний колір.

Після цього отриманий результат візуалізації можна зберегти як зображення у форматах .png та .jpeg, відправити по електронній пошті або зберегти посилання на результат.

Другий аналог – Microsoft Excel – комп'ютерна багатофункціональна програма від корпорації Microsoft для роботи з таблицями, проведення розрахунків, аналізу даних, прогнозування, складання графіків та інше. Програма приймає дані тільки у табличному вигляді, тобто файли у форматі .csv, .xls. В Microsoft Excel є можливість проводити розрахунки над даними і відобразити результат на діаграмі чи графіку, проте кількість можливих функцій обмежена. Кастомізація візуалізації даних досить різноманітна – можна задати колір, розмір, назви осей, назви стовпців тощо. Даний аналог надає користувачу більш різноманітний вибір діаграм, графіків користувачу, порівняно з першим аналогом. На рис.1.2 ми можемо побачити усі види діаграм та графіків, які доступні користувачу у програмі Microsoft Excel:

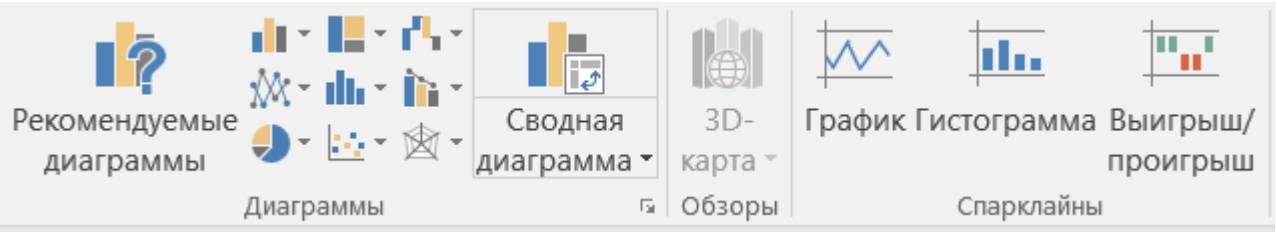

Рисунок 1.2 – Види діаграм, графіків, доступних користувачу в Microsoft Excel

Як можемо побачити, в цій програмі також можна візуалізувати дані у вигляді гістограм, кругових діаграм, діаграм розсіювання, блочних діаграм, лінійних графіків, павука тощо.

Також Microsoft Excel має таку функціональність як «Рекомендована діаграма», яка на основі введених користувачем даних пропонує найбільш підходящий тип діаграми. Для одного набору даних рекомендованих діаграм може бути декілька. Дана програма використовується зазвичай для складних розрахунків, звітів тощо.

Третій аналог – RawGraphs – онлайн програма для візуалізації графіків. Коштує 45$ на місяць. Дуже схожа на перший аналог, проте відрізняться від нього тим, що надає набагато ширший способів візуалізації даних у вигляді графіків, діаграм і т.д та більшу функціональність для кастомізації візуалізації даних. Наприклад, окрім можливості зміни таких стандартних параметрів як колір, розмір, назви осей, назви стовпців, можна конфігурувати відступи, колір заднього фону. На рис.1.3 відображено усі доступні способи візуалізації даних у програмі RawGraphs:

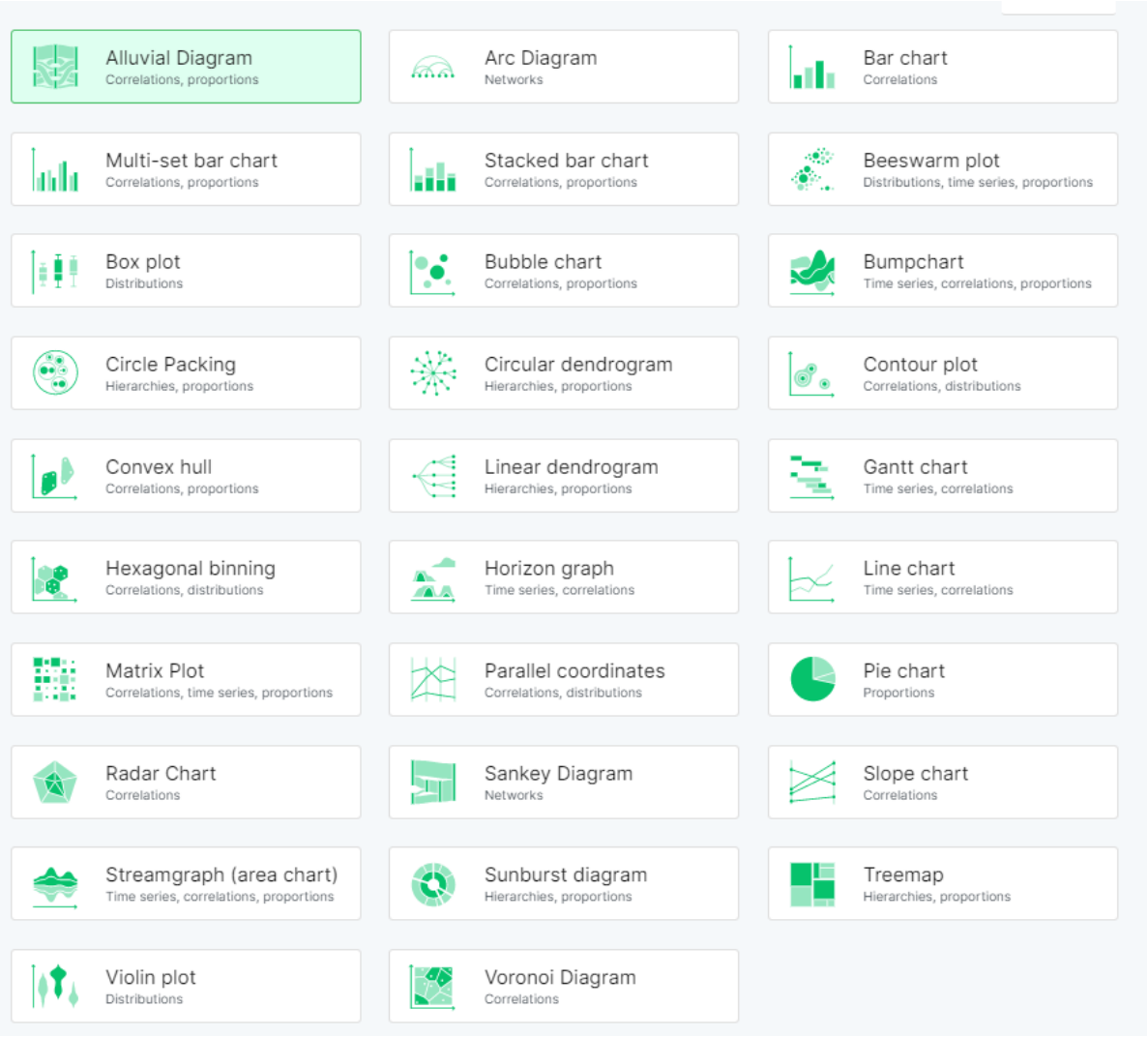

Рисунок 1.3 - Способи візуалізації даних у програмі RawGraphs

Готовий результат також можна зберегти як зображення або зберегти посилання на графік/діаграму. Слід зазначити, що RawGraphs – програма з відкритим кодом початковим кодом, що дає можливість іншим розробникам розширювати існуючий функціонал програми для свої бізнес-потреб.

Visu – безкоштовний програмний засіб для візуального відображення властивостей освітньої програми. Користувач вводить дані у форматі .csv файлу і має можливість візуалізувати дані у вигляді діаграм. Результат не зберігається і кастомізація відсутня. Кожне відображення даних засновано на відповідному розрахунку. Наприклад, для діаграми, що відображає результати навчання для кожної дисципліни, відображаються по осі абсцис усі можливі результати навчання, а по осі ординат – кількість дисциплін, які містять вказані результати навчання. В даному програмному засобі візуалізація даних доступна у трьох видах діаграм: стовпчастої діаграми, горизонтальної діаграми, у вигляді павука. Усі перераховані види можна побачити на рис.1.4:

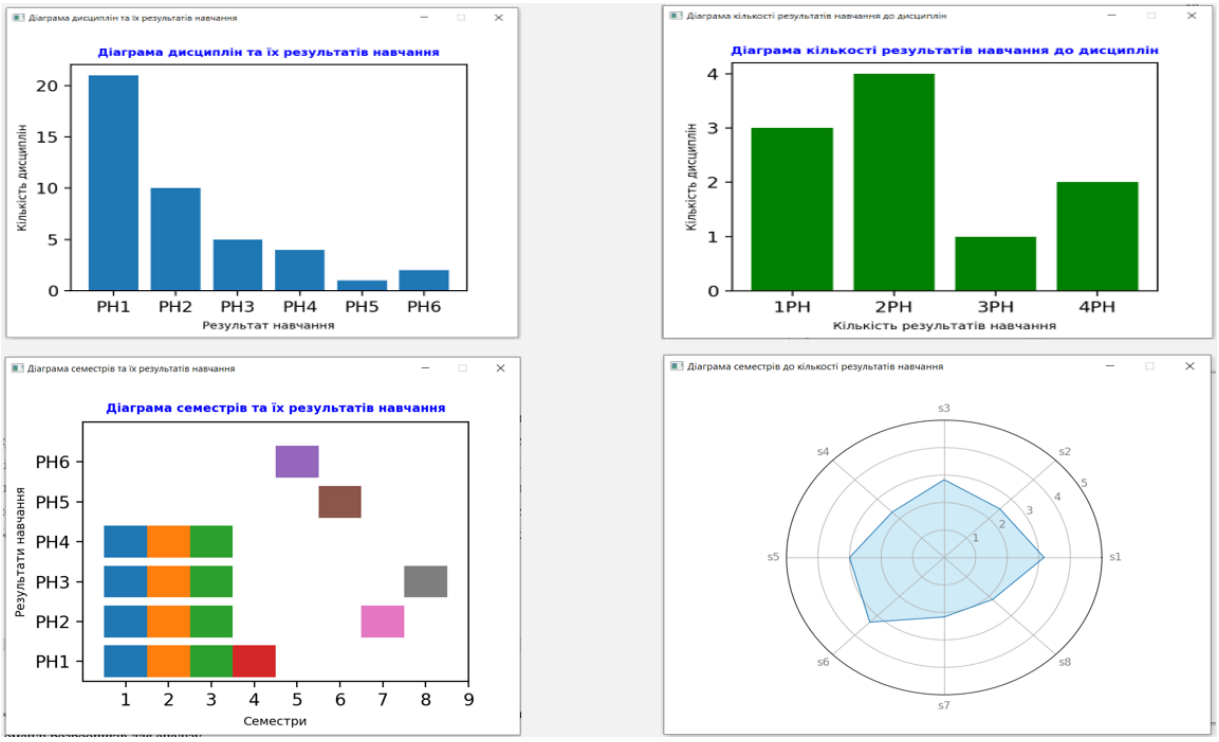

Рисунок 1.4 – Види діаграм у програмному засобі Visu

Слід зазначити, що програмний засіб Visu створений для візуалізації не будь-яких даних, а тільки тих, що описують властивості освітньої програми.

Як висновок, можемо зазначити, що у всіх програмних системах, наведених у списку аналогів, є як переваги, так і недоліки, в порівнянні з розробленим програмним засобом. ChartBlocks та RawGraphs пропонують платний функціонал, на відміну від Microsoft Excel та Visu. У програмі RawGraphs найбільший вибір способів візуалізації даних серед усіх трьох аналогів: дані можна представити у вигляді лінійних графіків, гістограм, діаграм розсіювання, кругових діаграм, матрицях, бульбашкових діаграм, горизонтальних діаграм, контурних діаграм тощо. Також ця програма надає багато можливостей з кастомізації вже створених діаграм. Окрім того, кожний аналог дозволяє користувачу завантажувати до програми власний файл з даними, а не вводити усю необхідну користувачеві інформацію до програми вручну, хоча функціонал з ручного вводу даних також присутній у двох аналогах: Microsoft Excel та ChartBlocks.

Переваги програмного засобу Visu перед своїми аналогами в тому, що він безкоштовний, підтримує файлове завантаження даних, а також вміє проводити розрахунки перед відображенням кінцевого результату у вигляді діаграми користувачу на екран. Останній пункт є найбільшою перевагою перед трьома аналогами, вказаними в таблиці. Наприклад, Visu вміє порахувати з внесених даних кількість дисциплін, які містять в собі конкретні кількості результатів навчання, у той час як у вказаних аналогів відсутня дана функціональність.

Окрім переваг, програмний засіб Visu має і ряд недоліків, а саме відсутність можливості зберегти кінцевий результат візуалізації у вигляді зображення, чи відправити його на електронну пошту. Також у Visu неможливо зробити кастомізацію діаграм, тобто змінити якісь властивості, як, наприклад, колір діаграми, розмір, надписи для осей, на відміну від усіх трьох аналогів, наведених у табл. 1.1.

2 МЕТОДИКА ПРОВЕДЕННЯ ДОСЛІДЖЕНЬ

Для того, щоб досягти мети роботи, а саме зменшити кількість годин для аналізу властивостей освітньої програми за допомогою програмного засобу, треба ретельно вибрати, в якому вигляді будуть зберігатися вхідні дані для діаграм у базі даних, завантажені користувачем до системи, і скільки часу піде на обробку цих даних – від цього залежить продуктивність програмного засобу.

Існує багато способів зберігання даних, як найпростіших, так і зі складною логікою та розширеною функціональністю для великих і комплексних систем.

Найпростішим способом зберігання даних є текстовий файл, проте він орієнтований на роботу тільки з невеликими об’ємами інформації. Інформація зберігається в будь-якому табличному вигляді (xls, csv) або строковому, але просто розділена між собою спеціальними знаками (кома, крапка з комою, пробіл тощо). Окрім того, цей спосіб має багато недоліків: відсутність паралельної обробки даних, складність встановлення зв’язків між компонентами даних, практичність тільки для систем з мінімальними вимогами читання/запису даних. Зазвичай, в текстовому вигляді зберігають конфігурації для систем або системних модулів.

Всі інші наступні засоби зберігання даних, які ми будемо розглядати, діляться на реляційні та нереляційні. Реляційні бази даних зберігають дані у таблицях, які можуть бути пов’язані між собою різними видами відношень: один до одного, багато до одного, багато до багатьох. Кожен стовпець у таблиці має ім'я та тип. Кожен рядок представляє окремий запис або елемент даних у таблиці, що містить значення для кожного зі стовпців. Як наслідок, забезпечується адаптованість до різних типів даних. Кожна таблиця представляє собою умовний об’єкт, у якого кожний стовпчик описує один атрибут об’єкту, а кожний рядок - відповідає конкретному екземпляру об’єкту. Також для доступу до даних використовується багатофункціональна мова структурованих запитів (SQL). Проте слід зазначити, що існують певні обмеження та

правила для реляційних баз, які допомагають зберігати цілісність та консистентність даних.

Розглянемо детальніше вид нереляційного способу зберігання даних – ієрархічні бази. В таких базах даних кожен запис має одного «предка». Це створює деревоподібну структуру, в якій записи класифікуються за їхніми відносинами з ланцюжком батьківських записів. Як наслідок, кожен запис може мати не більше одного предка, зв'язки між записами виконані у вигляді фізичних покажчиків і неможливо реалізувати відносини «багато-до-багатьох». Такий спосіб нереляційного зберігання даних широко використовуються у файлових системах, адже один файл, має тільки одного предка – папку, в якій він знаходиться. В ієрархічних базах легко досягти потрібного рівня безпеки, адже достатньо тільки встановити відповідні права на конкретний файл чи папку.

Наступним способом нереляційного зберігання є мережеві бази. В них між таблицями та записами можуть бути декілька різних зв’язків, і база даних зберігає усі зв’язки і необхідну інформацію для цих зв’язків у спеціальні індексні файли, тому не потрібно витрачати час на пошук потрібних даних.

Ще один спосіб нереляційної організації даних – NoSQL. Для зберігання інформації використовується структура «ключ-значення», коли надається ключ та об'єкт даних, який потрібно зберегти, по цьому ключу. Щоб запросити дані, треба відправити ключ і отримати blob-об’єкт (масив двійкових даних). Даний спосіб має як свої переваги, так і недоліки. Перевагами є те, що такі сховища забезпечують швидкий та маловитратний доступ, немає жорсткої схеми відношення між даними, тобто наявна можливість зберігати одночасно різні типи даних. Недоліками вважаються відповідальність розробника за визначення схеми іменування ключів і відповідність даних конкретному типу чи формату.

Для програмного засобу Visu збереження даних у текстовому файлі не є доцільним, оскільки об’єм інформації планується бути великим, між записами повинен існувати зв’язок (зв’язок між дисципліною та результатом навчання), а в

текстовому файлі такі зв’язки не можна визначити. Оскільки між дисципліною та результатом навчання існує зв’язок «багато до багатьох», ієрархічна організація даних також не підходить для програмного засобу, бо в ній відсутня підтримка такого зв’язку – у одного запису «предків» може бути багато. Мережева організація даних також не підходить, тому що між властивостями освітньої програми (дисциплінами та результатами навчання) відсутні різноманітні зв’язки, є лише один – «дисципліна має результат навчання». Для NoSQL, основа якого полягає в збереженні даних у вигляді «ключ-значення», приблизна структура виглядала би таким чином:

```
{"discipline1":" learningoutcomelist":["learning outcome1","learning
outcome2","learning outcome3"], semester:1}
"discipline2": "learningoutcomelist":["learning outcome2","learning
outcome4","learning outcome5"], semester:1}
"discipline3": "learningoutcomelist":["learning outcome1","learning
outcome5"], semester:2}
"discipline4": "learningoutcomelist":["learning outcome1","learning
outcome2","learning outcome3"], semester:3}
```

Це не є ефективною організацією даних для такої структури, де дані мають відношення «багато до багатьох», тому що для того, щоб знайти всі дисципліни, в яких є конкретний результат навчання, треба перебрати для ідентифікатора результату навчання усі масиви з дисциплінами, що матиме не тільки негативний наслідок на продуктивність такого запиту, а зробить програмне написання запиту в рази важчим.

Як результат, для реалізації відношення «багато до багатьох» між властивостями освітньої програми найкраще підходить саме реляційна організація даних, де зв’язок між двома сутностями-таблицями просто перетвориться у створення додаткової таблиці, яка не тільки зможе зберігати посилання на кожну з сутностей, а й зберігати додаткові параметри, як наприклад семестр.

3 ВИЗНАЧЕННЯ ВИМОГ ДО ПРОГРАМНОГО ЗАСОБУ

Одним з перших та надзвичайно важливих етапів під час створення будь-якого програмного продукту є технічне завдання. Технічне завдання – це документ, у якому викладені усі вимоги (функціональні, нефункціональні, апаратні, програмні), параметри і основні характеристики програмного продукту [4]. Також до складу цього документа може входити список вимог, що стосуються тестування продукту. Він укладається між замовником і виконавцем, і від детального та правильного опису технічного завдання залежить успіх розробки програмного продукту, що в кінці зможе цілком вдовольнити замовника.

До складу технічного завдання для проектування та розробки програмного засобу Visu увійдуть функціональні вимоги у вигляді діаграми та сценаріїв варіантів використання, нефункціональні вимоги у вигляді сценаріїв, а також апаратні та програмні вимоги.

3.1 Діаграма варіантів використання

В першу чергу слід зазначити, що варіант використання або прецедент – це опис того, що програмний продукт повинен робити. Тобто усю функціональність системи можна описати у вигляді прецедентів. Виконавцями прецедентів називають акторами і їх можуть бути багато. Графічне зображення взаємодії акторів і прецедентів у рамках однієї системи називають діаграмою варіантів використання або діаграмою прецедентів.

На рис. 3.1 представлена діаграма варіантів використання для програмного засобу Visu:

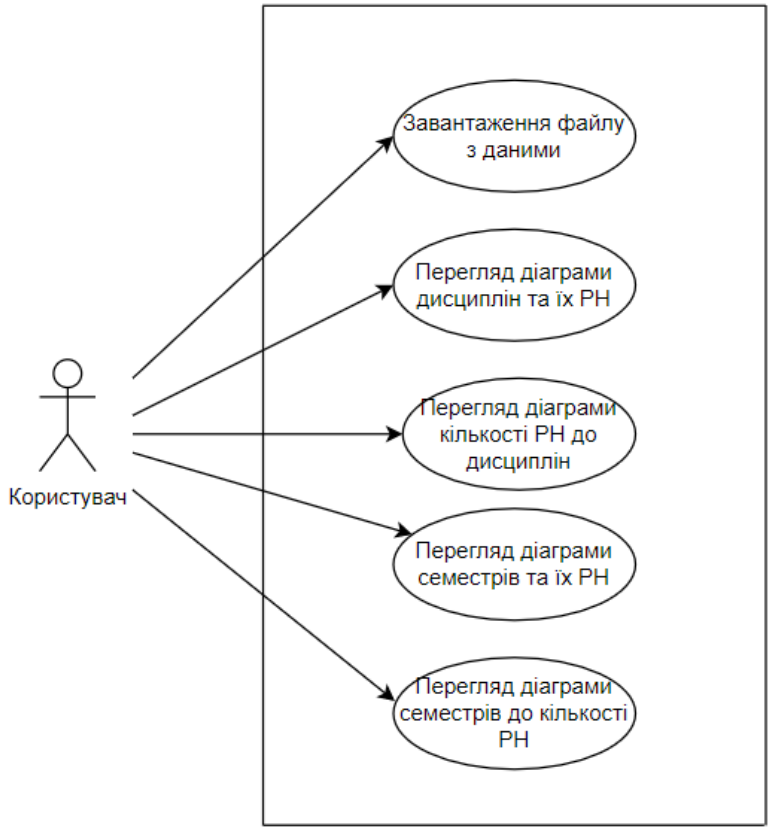

Рисунок 3.1 – Діаграма варіантів використання для програмного засобу Visu

На даний момент розроблюваний програмний засіб передбачає одного актора – Користувача, яким може виступати як укладач освітньої програми, так і особа, що ухвалює освітню програму. У Користувача є п'ять можливостей:

а) завантаження файлу з даними, що відображають властивості освітньої програми – дисципліни та результати навчання (РН);

б) перегляд діаграми усіх можливих дисциплін та РН, що входять до складу цих дисциплін;

в) перегляд діаграми кількості РН та дисциплін, що містять цю кількість;

г) перегляд діаграми семестрів та усіх можливих відповідних РН цих дисциплін;

ґ) перегляд діаграми семестрів до кількості відповідних РН.

Для більш детального опису кожного з варіантів використання використовують сценарії варіантів використання.

3.2 Сценарії варіантів використання

Опишемо детальніше кожен з варіантів використання програмного засобу Visu у вигляді сценаріїв.

Опис варіанту використання «Завантаження файлу з даними»

Мета: виклик даної функції завантажує файл з даними до системи.

Актори: користувач.

Тригер: користувач відкрив програму.

Основний сценарій:

1. Користувач перейшов на вкладку «Конфігурація».

2. Система відкриває нове вікно конфігурації с повідомленням про те, що користувачу потрібно завантажити файл до системи у форматі .csv та кнопкою «Завантажити файл».

3. Користувач викликає функцію «Завантажити файл».

4. Система відкриває у новому вікні файловий провідник комп’ютера.

5. Користувач обирає файл у форматі .csv та викликає у файловому провіднику функцію «Відкрити».

6. Система виводить на екран повідомлення про успішне завантаження файлу.

7. Користувач закриває вікно конфігурації.

Опис варіанту використання «Перегляд діаграми дисциплін та їх РН»

Опис: виклик даної функції відображає діаграми дисциплін та їх РН у вигляді гістограми.

Актори: користувач.

Передумова: користувач відкрив програму.

Тригер: користувач перейшов на вкладку «Діаграми».

Основний сценарій:

1. Система відображає користувачу на екрані чотири доступні функції: «Діаграма дисциплін та їх результатів навчання», «Діаграма кількості результатів навчання до дисциплін», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів до кількості результатів навчання» відповідно.

2. Користувач вибирає функцію «Діаграма дисциплін та їх результатів навчання».

3. Система відкриває нове вікно з зображенням гістограми, де по осі абсцис знаходяться усі можливі результати навчання, а по осі ординат – кількість дисциплін, які містять вказані результати навчання.

4. Користувач закриває вікно.

Опис варіанту використання «Перегляд діаграми кількості результатів навчання до дисциплін»

Опис: виклик даної функції відображає діаграму кількості результатів навчання до дисциплін у вигляді гістограми.

Актори: користувач.

Передумова: користувач відкрив програму.

Тригер: користувач перейшов на вкладку «Діаграми».

Основний сценарій:

1. Система відображає користувачу на екрані чотири доступні функції: «Діаграма дисциплін та їх результатів навчання», «Діаграма кількості результатів навчання до дисциплін», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів до кількості результатів навчання» відповідно.

2. Користувач викликає функцію «Діаграма кількості результатів навчання до дисциплін».

3. Система відкриває нове вікно з зображенням гістограми, де по осі абсцис знаходиться кількість результатів навчання, а по осі ординат – кількість дисциплін, які містять вказану кількість результатів навчання.

4. Користувач закриває вікно.

Опис варіанту використання «Діаграма семестрів та їх результатів навчання»

Опис: виклик даної функції відображає діаграму семестрів та їх результатів навчання у вигляді горизонтальної діаграми.

Актори: користувач.

Передумова: користувач відкрив програму.

Тригер: користувач перейшов на вкладку «Діаграми».

Основний сценарій:

1. Система відображає користувачу на екрані чотири доступні функції: «Діаграма дисциплін та їх результатів навчання», «Діаграма кількості результатів навчання до дисциплін», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів до кількості результатів навчання» відповідно.

2. Користувач викликає функцію «Діаграма семестрів та їх результатів навчання».

3. Система відкриває нове вікно з зображенням горизонтальної діаграми, де по осі абсцис відображені семестри, а по осі ординат результати навчання, які були в кожному семестрі.

4. Користувач закриває вікно.

Опис варіанту використання «Перегляд діаграми семестрів до кількості результатів навчання»

Опис: виклик даної функції відображає діаграму семестрів до кількості результатів навчання у вигляді павука.

Актори: користувач.

Передумова: користувач відкрив програму.

Тригер: користувач перейшов на вкладку «Діаграми».

Основний сценарій:

1. Система відображає користувачу на екрані чотири доступні функції: «Діаграма дисциплін та їх результатів навчання», «Діаграма кількості результатів навчання до дисциплін», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів до кількості результатів навчання» відповідно.

2. Користувач викликає функцію «Діаграма семестрів до кількості результатів навчання».

3. Система відкриває нове вікно з зображенням павука, що відображає скільки результатів навчання є в кожному семестрі.

4. Користувач закриває вікно.

У даному підрозділі було детально описано кожен варіант використання з діаграми варіантів використання (рис. 3.1) у вигляді сценарію, що складається з короткого опису варіанту використання, передумови, тригеру для запуска варіанту використання та основного сценарію.

3.3 Нефункціональні вимоги

Нефункціональні вимоги представляють вимоги до програмного продукту, які задають критерії для оцінки якості його роботи. На відміну від функціональних вимог, які визначають, що система повинна робити, нефункціональні вимоги визначають якою система повинна бути [5].

Нефункціональні вимоги програмного засобу Visu описані чотирма критеріями: надійність, зручність використання, безпека, продуктивність. Нижче наведені сценарії нефункціональних вимог для кожного критерію:

а) надійність: завантажені користувачем дані, які необхідно зберегти, система зберігає не менше ніж у 98%;

б) зручність використання: кількість кліків користувача, що прагне завантажити файл до системи, складає не більше 3; кількість кліків користувача, що прагне переглянути графік або діаграму, складає не більше 2;

в) продуктивність: час відображення графіку/діаграми на екрані складає не більше 2 сек;

г) безпека: система захищає дані користувачів від порушення цілісності інформації не менше ніж у 98% випадків.

3.4 Апаратні та програмні вимоги

Апаратні вимоги – це характеристики, яким потрібен відповідати цифровий пристрій для нормальної роботи програмних систем, які знаходяться на цьому пристрої. Такі вимоги можуть бути мінімальними та рекомендованими. Мінімальні означають, що цифровий пристрій буде працювати при таких характеристиках, проте повільно. Рекомендовані вимоги забезпечують максимально комфортні умови для роботи відповідного програмного забезпечення на цьому цифровому пристрої.

Програмний засіб Visu потребує комп'ютер із процесором або аналогічним Pentium III тактовою частотою 600 МГц або вище (рекомендується 1 ГГц або вище). Мінімальний обсяг оперативної пам'яті 192 Мб (рекомендується 512 Мб або більше). 200 Мб вільного дискового простору. Завантаження інтерпретатору для виконання скриптів на мові Python – CPython, реляційної бази даних PostgreSQL.

Апаратні та програмні вимоги є не менш важливими за функціональні та нефункціональні вимоги для розробки будь-якої програмної системи, адже якщо не вистачить ресурсів для запуску програмної системи, то допуск до її функціоналу тим паче буде відсутній.

4 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

Проектування програмного продукту виконується після складання технічного завдання і є не менш важливим етапом у життєвому циклі розробки програмного продукту. Під час проектування визначають архітектуру, компоненти та програмні інтерфейси [6].

4.1 Архітектура програмного засобу

Архітектура (software architecture) – це структура програми, яка включає програмні компоненти, видимі зовні властивості цих компонентів, а також відносини між ними. В архітектурі описано саме організацію програмних компонентів: їх кількість, якість, протоколи взаємодії між компонентами. Саме від архітектури залежить собівартість програмного продукту та можливість повторного використання коду, що в майбутньому може зекономити час і гроші власнику програмного продукту.

Якісно розроблена архітектура повинна відповідати п’яти принципам [7]:

а) принципу відкриття-закриття (Open Close Principle або OCP) – коли програмні компоненти, такі як модулі, класи, методи, повинні бути відкриті для розширення, але закриті для змін;

б) принципу Заміщення Лісков (Liskov's Substitution Principle) – коли класи, що розширюють базовий функціонал, можуть бути заміщені базовими класами, і це не приведе до зміни поведінки базових класів, адже ідея полягає саме в розширенні базового функціоналу, а не його тотальній заміні;

в) принципу Єдиної Відповідальності (Single Responsibility Principle) – коли один клас відповідальний тільки за одну функціональність. Наприклад, для класу, що створений для роботи з вводом-виводом інформації у файли, недоцільно додавати функціональність для вводу-виводу інформації в мережу інтернет. В такому випадку

правильним рішенням є створення двох різних класів, кожен з яких буде відповідати за свою роботу з інформацією;

г) принципу Відділення Інтерфейсу (Interface Segregation Principle) – коли клієнти не повинні бути залежними від інтерфейсів, які вони не використовують. Тобто коли створюються інтерфейси, треба додавати тільки ті методи, які там повинні бути, інакше інтерфейс з зайвими методами буде вважатиметься «забрудненим»;

д) принципу інверсії залежностей (Dependency Inversion Principle) – коли залежності між компонентами всередині системи базуються на основі абстракцій, причому абстракції нижчих рівнів системи не повинні залежати від абстракцій верхніх рівнів.

За довгі роки розвитку програмного забезпечення програмістам вдалось винайти надійні підходи для усунення проблем та недоліків проектування забезпечення з недосконалою архітектурою або її відсутністю [8]. Нижче перечислимо чотири найвідоміших та найефективніших види архітектури програмного забезпечення та обґрунтуємо свій вибір архітектури при проектуванні програмного засобу Visu.

Перший вид – багатошарова архітектура (Layered Architecture). Її принцип роботи полягає в тому, що програмна система розділена на логічні шари, які лежать один на одному, і кожен з них відповідальний за свою задачу. Як правило, розділення системи відбувається на три шари: шар представлення, шар бізнес-логіки та шар передачі даних. Шар представлення відповідає за графічний інтерфейс користувача. Шар бізнес-логіки відповідає за саму логіку програмної системи, яка при цьому може змінюватися в залежності від змін бізнес-вимог, і не впливати при цьому на інші шари. Шар передачі даних відповідає за обмін даними між програмною системою та сховищем даних та інколи може обробляти при цьому дані, що при цьому ніяк не впливатимуть на бізнес-логіку програмної системи.

Такий вид архітектури має як свої недоліки, так і переваги. Серед переваг можна виділити те, що така архітектура пропонує абстракцію завдяки розділенню відповідальностей між шарами; ізоляція кожного шару забезпечує мінімальний вплив

або його відсутність на інші шари; підвищує керованість програмного забезпечення завдяки слабкій пов’язаності. Недоліки такого виду архітектури в тому, що вона не пропонує велику масштабованості і, окрім того, дані системи обов’язково повинні проходити через усі шари архітектури, навіть якщо в цьому немає потреби.

Другий вид архітектури, що широко використовується при проектуванні програмних систем, є багаторівнева архітектура (Tiered Architecture). Цей підхід поділяє програмну систему на декілька рівнів, де обов’язково є взаємодія «клієнт – сервер». Така архітектура може мати від одного до багатьох рівнів, що розділяють відповідальності між постачальником даних та їх споживачем. В якості клієнта виступають рівні представлення, бізнес-логіки та передачі даних, а в якості сервера – сховища та бази даних. Для зв’язку між рівнями використовується шаблон – «запит - відповідь». Така побудова дає можливості для горизонтальної та вертикальної масштабованості, на відміну від багатошарової архітектури. Реалізація багаторівневої архітектури дорожча за інші види архітектур, проте забезпечує найвищу продуктивність. Через це її доцільно використовувати для великих та комплексних програмних систем.

Третій вид архітектури є сервіс-орієнтована архітектура (Service Oriented Architecture – SOA). Така модель складається з компонентів, які взаємодіють між собою через чітко зазначені сервіси. Модель складається з п’яти елементів: сервісу, сервісної шини, сервісного репозиторію, елементу безпеки та елементу управління. Принцип роботи сервіс-орієнтованої архітектури такий: клієнт надсилає запит по стандартному протоколу та у конкретному форматі даних через мережу. Цей запит обробляється сервісною шиною, що вважається основою сервіс-орієнтованої архітектури та відповідає за оркестрування та маршрутизацію. Сервісна шина за допомогою сервісного репозиторію надсилає запит до спеціального сервісу, який може взаємодіяти з іншими сервісами та базами даних, щоб сформувати відповідь. Повний виклик відповіді на запит узгоджується з елементами керування та безпеки

для виконання безпечної та коректної транзакції. Сервіси можуть бути двох видів: атомарного та композиційного.

Четвертим видом архітектури вважається мікросервісна архітектура (Microservice Architecture). Така модель складається з набору невеликих сервісів, кожен з яких працює у власному процесі, виконує власну задачу. Централізоване керування такими сервісами мінімальне, тому сервіси можуть бути написані на різних мовах, мати різну структуру, використовувати різні технології збереження та обробки даних. Сервіси не залежать один від одного, тобто зміни в одному сервісі не матимуть жодного впливу на інші. Розгортання таких сервісів також автономне. Переваги мікросервісної архітектури в тому, що вона забезпечує модульність системи, слабку зв’язаність між сервісами, а отже високу гнучкість та масштабованість. Недоліки в тому, що великою кількістю сервісів важко керувати. Окрім того, у такої архітектури з’являються такі проблеми як затримки в мережі, балансування навантаження, необхідність тестування кожного сервісу окремо. Також існує підвищений ризик збою при обміні даними між сервісами.

Для програмного засобу Visu була вибрана багатошарова архітектура, оскільки:

а) умовно систему Visu можна поділити на три логічні шари – шар представлення, шар бізнес-логіки та шар передачі даних, і при роботі системи дані повинні проходити через усі три, адже користувач через графічний інструмент запитує дані, система в шарі бізнес-логіки оброблює запит користувача і визначає, які самі дані потрібні для формування відповіді, а в шарі передачі даних запит користувача перетворюється у відповідний запит для бази даних;

б) програмний засіб Visu не є комплексною системою, і не потребує великих ресурсів для складних обчислень, тому багаторівнева архітектура не доцільна;

в) Visu не потребує різноманітних сервісів для обчислення та формування відповіді користувачу, оскільки в системі мета однакова – представити користувачу діаграму на основі того чи іншого запиту.

Багатошарова архітектура програмного засобу Visu представлена на рис. 4.1

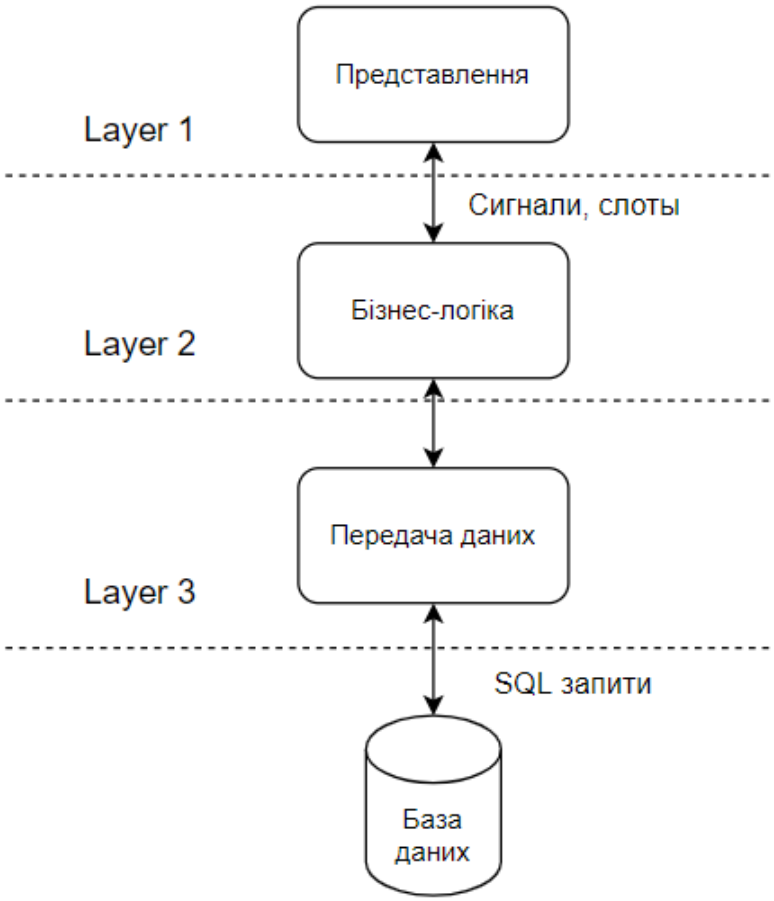

Рисунок 4.1 – Багатошарова архітектура програмного засобу Visu

Як результат, архітектура програмного засобу складається з трьох шарів: Представлення, Бізнес-логіки, Передачі даних та Бази даних.

4.2 Технології розробки програмного засобу

Програмний засіб написаний на мові програмування Python через ряд причин. По-перше, Python – високорівнева мова програмування загального призначення. Вона є об’єктно-орієнтованою і мультипарадигмальною, тобто підтримує імперативне, процедурне, структурне і функціональне програмування. По-друге, еталонною реалізацією цієї мови є інтерпретатор CPython, який компілює вихідні тексти у

високорівневий байт-код, що виконується у стековій віртуальній машині. Це забезпечує високу продуктивність та кросплатформеність. Також особливістю цієї мови програмування в тому, що структура розбита на пакети, які встановлюються тільки тоді, коли є потреба в функціональності цих пакетів.

Шар Представлення, що відповідає за графічний інтерфейс користувача (вікна, діалогові вікна, кнопки, поля для вводу тексту, вкладки), в Visu реалізований за допомогою бібліотеки PyQt, яка є розширенням фреймворку графічних інтерфейсів Qt для мови програмування Python. PyQt предоставляє великий вибір функціоналу, як наприклад:

а) набір віджетів графічного інтерфейсу;

б) стилі віджетів;

в) доступ до баз даних за допомогою SQL (ODBC, MySQL, PostgreSQL, Oracle);

г) підтримку інтернаціоналізації (i18n);

д) парсер XML;

е) підтримку SVG;

є) інтеграцію з WebKit, двигуном рендерингу HTML;

ж) підтримку відтворення відео та аудіо.

Для надання функціональності елементам графічного інтерфейсу використовуються події. У моделі подій є три учасники: джерело події, об'єкт події, мета події. Джерело події – це об'єкт, стан якого змінюється. Він викликає подію. Подія інкапсулює зміну стану у джерелі події. Мета події – це об'єкт, якому потрібне повідомлення. Об'єкт джерела події делегує завдання обробки події цілі події. Для роботи з подіями PyQt5 має унікальний механізм сигналів та слотів. Сигнали та слоти використовуються для зв'язку між об'єктами. Сигнал спрацьовує тоді, коли відбувається конкретна подія. Слот може бути будь-якою функцією. Слот викликається, коли спрацьовує сигнал. Бібліотека невелика за розміром, але дуже функціональна.

Для зручності створення графічних елементів використовується графічний дизайнер Qt Creator. До його складу входить панель інструментів, на якій представлені усі доступні графічні елементи: вікна, кнопки, меню, віджети, діалогові вікна тощо. Для кожного елемента є набір властивостей, які можна змінювати за допомогою «Редактора властивостей». Розроблений інтерфейс зберігається в файлі з розширенням .ui, який потім програма pyuic перетворює в файл з кодом на мові Python.

В шарі Бізнес-логіки зосереджена основна логіка будування діаграм в залежності від запиту користувача, у вигляді програмних класів, а також функціонал по завантаженню конфігураційного файлу з даними про властивості освітньої програми у форматі .csv до системи та по обробці даних.

В шарі Передачі даних знаходиться логіка по створенню запитів до бази даних на мові структурованих запитів SQL (Structured Query Language), на основі яких вибираються дані з бази даних, необхідні для побудови діаграм. Як було заплановано в другому розділі, у програмному засобі використовується саме реляційна база даних. Для управління базою даних є багато систем управління (СУБД), кожна з яких має свої недоліки та переваги. Розглянемо детальніше найбільш популярні СУБД:

1. MySQL – вільна система управління базами даних, яка використовується в якості сервера, до якого звертаються як локальні так і віддалені клієнти. В цій системі гнучкість підтримується великою кількістю різних типів таблиць. MySQL має API (програмний інтерфейс системи) та конектори з великою кількістю мов програмування, серед яких і Python. MySQL – найкраще рішення для малих та середніх програмних систем.

2. Microsoft SQL Server – СУБД від компанії Microsoft, що використовує власну мову структурованих запитів Transact-SQL. Вона дозволяє підключитися локально чи по мережі до бази даних, відправити команду по спеціальному протоколу TDS (Tabular Data Stream) і отримати результат. Дана програмна система – це повноцінна клієнт-серверна архітектура, в якій обчислення проводяться на сервері бази даних, а

не на стороні клієнта, і по мережі не завантажуються самі таблиці представлення даних, а тільки результати виконаних запитів.

3. PostgreSQL – вільна об’єктно-реляційна СУБД, що базується на основі SQL, має необмежений розмір бази даних, необмежену кількість індексів для таблиці, максимальний розмір таблиці – 1Тбайт, а максимальний розмір поля таблиці – 1 Гбайт. Основними перевагами цієї системи вважають високопродуктивні і надійні механізми транзакцій та реплікацій, розширена система багатьох вбудованих мов програмування, наслідування, вбудована підтримка даних у форматі JSON, розширюваність. Проте найбільшою перевагою PostgreSQL є підтримка одночасної модифікації бази даних декількома користувачами за допомогою спеціального механізму MVCC (Multiversion Concurrency Control), завдяки якому відсутні блокування читання.

4. Oracle Database – об’єктно-реляційна СУБД від компанії Oracle. Система поставляється шести різних видів, для різних сценаріїв розробки та розгортування систем. Кожен вид надає власні набори властивостей СУБД, розмір пам'яті, кількість процесорів тощо. Перевагами системи є автономні транзакції, автоматичне керування збереженням файлів, наявність резервного серверу тощо. Oracle Database встановлюють для складних та великих програмних систем, які потребують підвищений вибір безпечності та продуктивності.

Для програмного засобу Visu була вибрана PostgreSQL через її переваги, а також наявності реалізацій для будь-яких платформ, як UNIX-систем, так і Microsoft Windows, що дасть можливість користувачам використовувати Visu на комп’ютері з будь-якою операційною системою.

Програмний код написаний у спеціальному середовищі розробки PyCharm. Це комерційне інтегроване середовище розробки для розробки програмного забезпечення на мові програмування Python, яке створене на основі вже відомого продукту Intellij Idea. Програма доступна на усіх відомих операційних системах. Щоб отримати для середовища додаткову функціональність, достатньо завантажити

потрібні плагіни, які є у вільному доступі в мережі інтернет. PyCharm має такі можливості як:

а) відладка коду за допомогою PyDev:

б) рефакторинг коду;

в) автодоповнення коду;

г) підтримка Git, Subversion, Mercurial та інших систем контролю версій.

Також це інтегроване середовище розробки дає можливість іншим розробникам писати власні плагіни, які будуть сумісні з PyCharm.

В якості клієнта до бази даних PostgreSQL використовується DBeaver – це клієнтська програма для реляційних баз даних та інструмент управління ними. Програма взаємодіє з базами за допомогою драйверів JDBC (англ. Java DataBase Connectivity). DBeaver зручний у використанні, оскільки має функції автозавершення коду та підсвічування синтаксису. Деякі синтаксичні помилки програма підсвічує ще на етапі написання запитів. Значною перевагою цієї програми також є можливість користувачу завантажувати плагіни для конкретних баз даних, що дозволяють змінювати більшу частину програми для підтримки специфічних функцій для необхідних баз даних. DBeaver випускається в двох версіях DBeaver Community Edition (DBeaver CE) та DBeaver Enterprise Edition (DBeaver EE). DBeaver CE є безкоштовною програмою та має відкритий вихідний код, у той час як DBeaver EE є платним. Платна версія має розширені можливості для роботи з noSQL (MongoDB і т.п.) базами даних. Для розробки та управління базою даних програмного засобу Visu була використана безкоштовна версія. Серед можливостей цієї версії є виконання запитів SQL, редактор даних, управління скриптами SQL, перегляд та редагування структури бази даних, імпорт, експорт та резервне копіювання баз даних. Також доступна функція генерації тестових даних для тестування бази даних.

4.3 Діаграми послідовності

Невід’ємною частиною проектування будь-якої програмної системи є використання UML (Unified Modeling Language) діаграм. Такі діаграми використовують для графічного позначення абстрактної моделі системи. UML була створена для визначення, візуалізації та документування програмних систем і може бути використана на всіх етапах життєвого циклу програм. Діаграма дозволяє представити програмну систему у такому вигляді, щоб потім її можна було легко перетворити у програмний код. В UML використовують 14 різних видів діаграм, але в даному підрозділі будуть представлені саме діаграми послідовності.

Діаграма послідовності – відображає взаємодію об'єктів в системі впорядкованих за часом. Зазвичай, такі діаграми відображають задіяні об’єкти та послідовність надісланих повідомлень [9]. Представимо на рис. 4.2 діаграму послідовності для завантаження користувачем файлу з даними до системи у конкретному форматі.

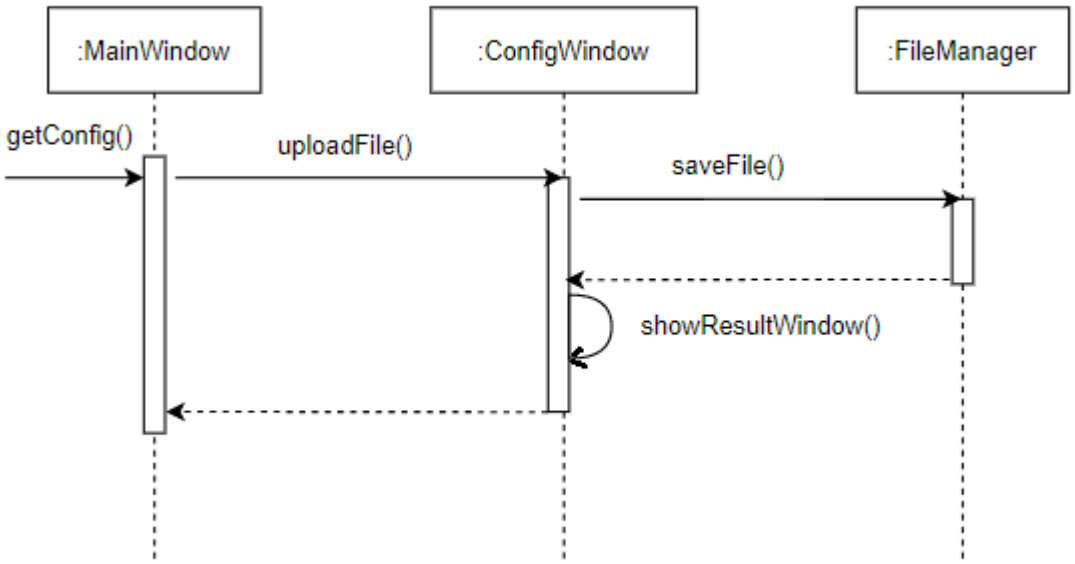

Рисунок 4.2 – Діаграма послідовності для завантаження файлу до системи Visu

На даній діаграмі послідовності зображено три головних об’єкта системи MainWindow, ConfigWindow, FileManager. MainWindow – об’єкт головного вікна, що містить вкладку «Конфігурація», ConfigWindow – об’єкт вікна конфігурації, який дозволяє користувачу загрузити файл та показує результат збереження файлу до системи, FileManager – об’єкт класу, який відповідає за завантаження файлу з даними та перетворення їх в запити SQL для вставки даних до бази даних. Основний сценарій виконання цієї функціональності такий: користувач запитує у головного вікна конфігураційне вікно за допомогою методу getConfig(), конфігураційне вікно надсилає запит до менеджера файлів зберегти файл у системі за допомогою метода saveFile(), після збереження файлі на конфігураційному вікні відображається повідомлення про результат збереження за допомогою методу showResultWindow().

Зобразимо на діаграмі послідовності (рис. 4.3) таку функціональність, як відображення користувачу діаграми дисциплін та їх результатів навчання.

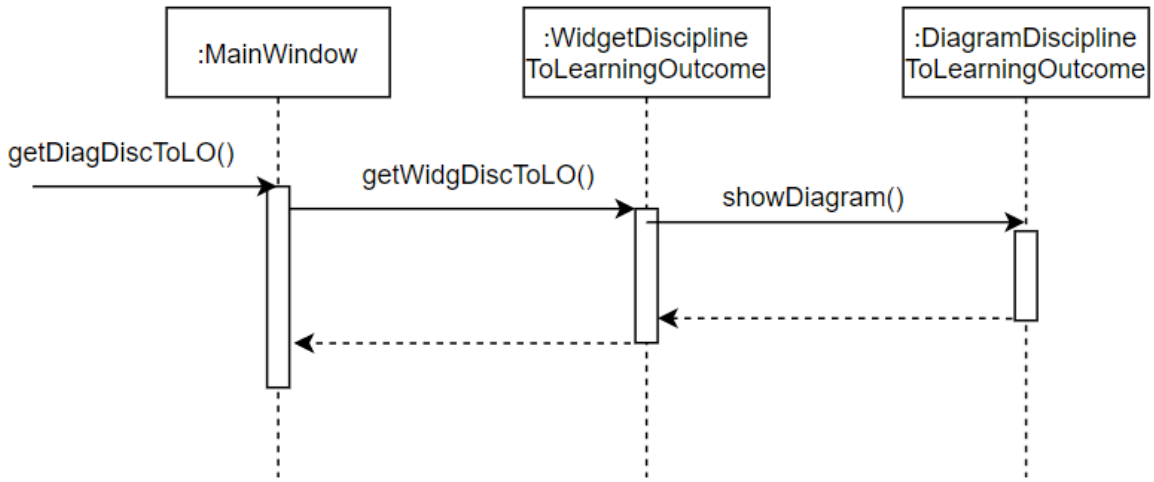

Рисунок 4.3 – Діаграма послідовності для відображення діаграми дисциплін та їх результатів навчання

Дана діаграма послідовності відображає процес взаємодії трьох основних об’єктів системи, що беруть участь в відображенні діаграми властивостей освітньої програми: MainWindow, WidgetDisciplineToLearningOutcome, DiagramDisciplineToLearningOutcome. MainWindow – головне вікно програмної системи, до якого звертається користувач для відображення вище зазначеної діаграми за допомогою метода getDiagDiscToLO(). Об’єкт головного вікна відправляє запит до об’єкта віджета діаграми відобразити потрібний віджет за допомогою методу getWidgDiscToLO(), об’єкт віджета відправляє запит до об’єкту діаграми за допомогою showDiagram() показати кінцеву діаграму дисциплін та їх результатів навчання.

Наступні діаграми послідовності описують такий самий сценарій отримання певної діаграми візуалізації освітньої програми, тільки запити від головного вікна програмного засобу відправляються різним об’єктам-віджетам і об’єктам-діаграмам.

На рис. 4.4 можна побачити діаграму послідовності для отримання користувачем діаграми відношення кількості результатів навчання до дисциплін.

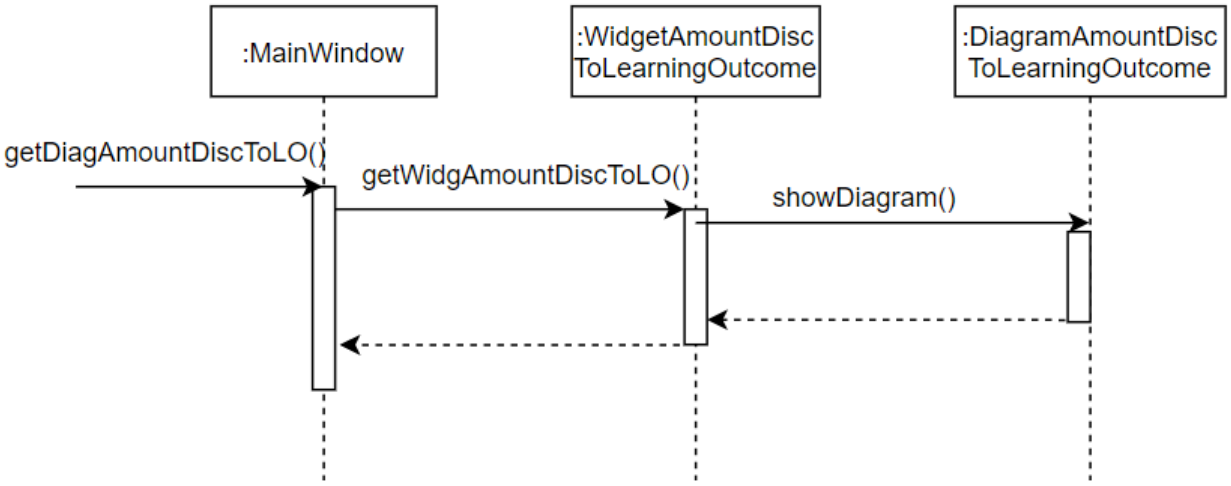

Рисунок 4.4 – Діаграма послідовності для відображення діаграми кількості результатів навчання до дисциплін

На рис. 4.5 можна побачити діаграму послідовності для отримання користувачем діаграми семестрів та результатів навчання.

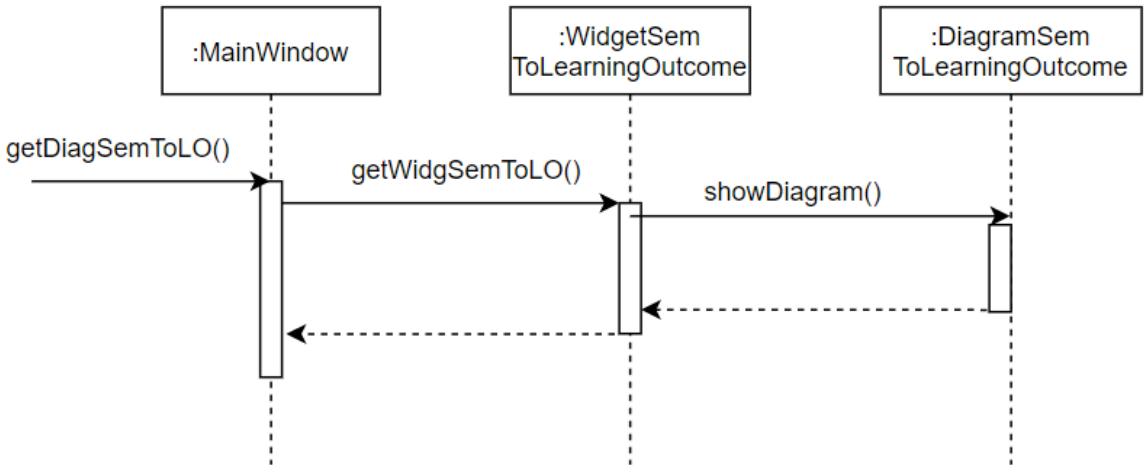

Рисунок 4.5 – Діаграма послідовності для відображення діаграми семестрів та результатів навчання

На рис. 4.6 можна побачити діаграму послідовності для отримання користувачем діаграми семестрів та кількості результатів навчання.

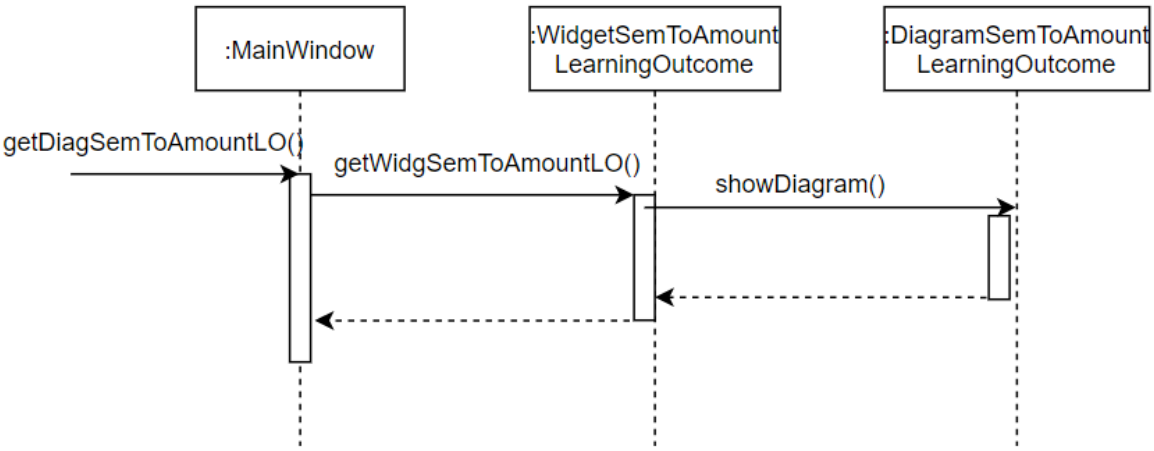

Рисунок 4.6 – Діаграма послідовності для відображення діаграми семестрів та кількості результатів навчання

У даному підрозділі була вибрана архітектура для програмного засобу Visu, технології для розробки засобу, а також були наведені п’ять діаграм послідовності для усіх варіантів використання системи.

4.4 Діаграма програмних класів

Для формалізації логічної моделі програмної системи на рівні структурних елементів, тобто класів, використовують діаграму програмних класів. В ній описані наявні в програмній системі класи, властивості класів та зв’язки між цими класами. Як відомо, кожний клас описує стан та поведінку об’єкта, на основі якого сам об’єкт створений. Стан об’єктів описується за допомогою атрибутів, а поведінка – за допомогою методів.

Всі наведені класи розміщені по відповідним пакетам. Для класів, пов’язаних з базою даних, створений пакет database. До нього входять класи Connection та Queries. Клас Connection відповідає за підключення до бази даних, і має такі методи як:

а) query() – відповідає за виконання запиту мовою SQL до бази даних;

б) query_with_param(param) – виконує запит до бази даних з динамічним параметром;

в) close() – відповідає за закриття зв’язку з базою даних;

г) cursor() –відповідає за завантаження даних, що являються результатом виконання запиту;

д) init(params) – створює зв’язок з базою даних, де в якості params передаються такі необхідні параметри як назва бази даних, до якої звертаються, хост на якому локально працює сервер СУБД, порт, користувач бази даних та пароль користувача.

На рис. 4.7 представимо діаграму класів програмного засобу Visu. Описані на діаграмі класи розміщені у різних пакетах для зручності їх використання.

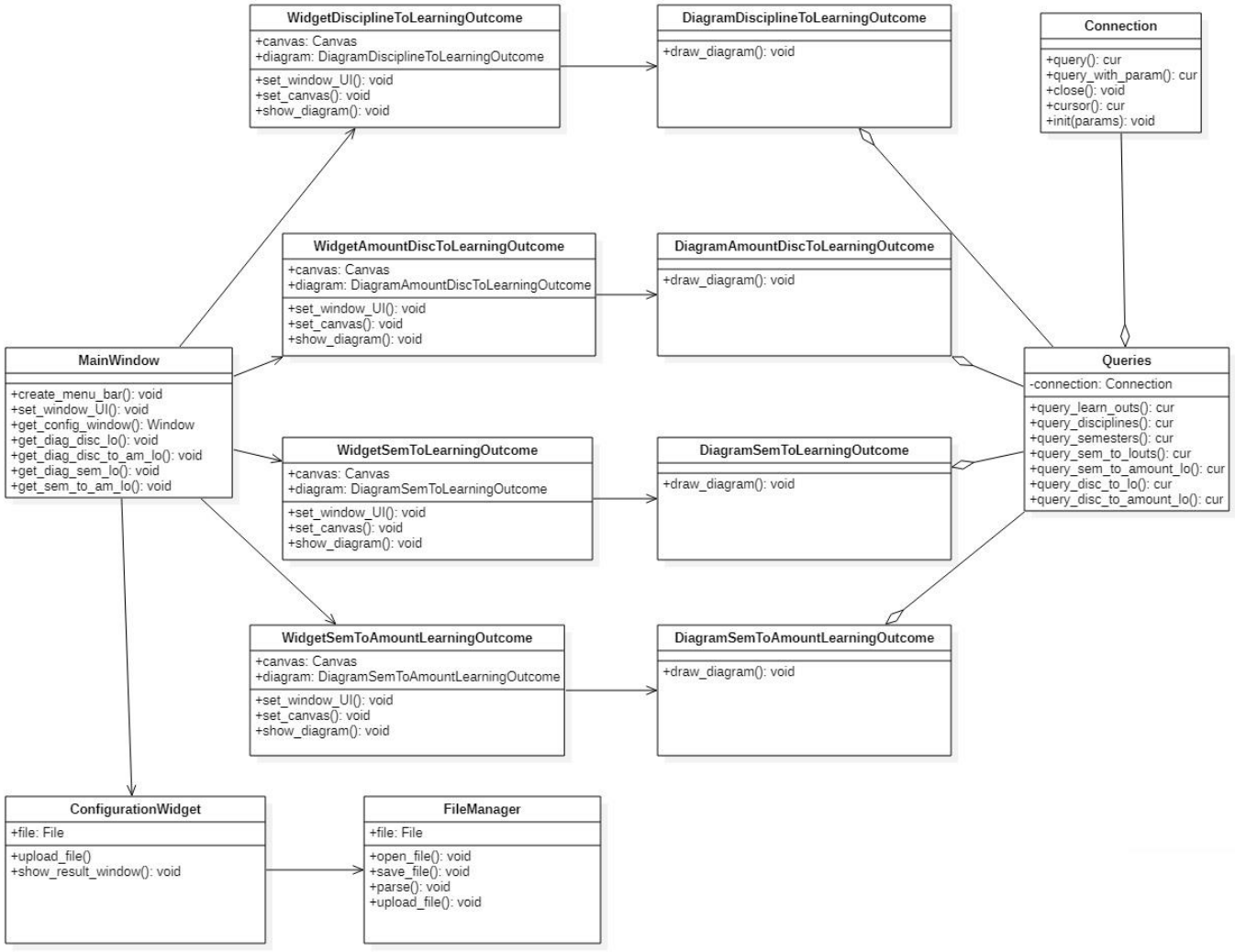

Рисунок 4.7 – Діаграма класів програмного засобу Visu

Наведемо детальний опис деяких програмних класів.

Клас Queries відповідає за формування конкретних запитів програмного засобу на мові структурних запитів SQL до бази даних. Клас Connection являється частиною цього класу, адже неможливо виконати запит до бази даних при цьому не ініціювавши зв'язок між програмною системою та базою, не завантажуючи кінцевих даних тощо. Queries має такі методи:

а) query_learn_outs() – запитує у бази даних всі результати навчання з їх параметрами id та name;

б) query_disciplines() – запитує у бази даних всі дисципліни з їх параметрами id та name;

в) query_semesters() – запитує у бази даних всі семестри;

г) query_sem_to_louts() – запитує у бази даних семестри та всі результати навчання, які були у відповідному семестрі;

ґ) query_sem_to_amount_lo() –запитує у бази даних семестри та загальну кількість результатів навчання, які були у відповідному семестрі;

д) query_disc_to_lo() – запитує у бази даних дисципліни та результати навчання, які були у відповідній дисципліні;

е) query_disc_to_amount_lo() – запитує у бази даних дисципліни та загальну кількість результатів навчання, які були у відповідній дисципліні;

є) query_learn_out_ids() – запитує у бази даних ідентифікатори усіх результатів навчання;

ж) query_learn_out_names() – запитує у бази даних імена усіх результатів навчання;

з) query_disc_amount() – представляє собою агрегуючу функцію, що формує загальну кількість дисциплін;

і) query_learn_out_amount() - представляє собою агрегуючу функцію, що формує загальну кількість результатів навчання і т.д.

У пакеті configuration знаходяться класи FileManager та ConfigurationWindow.

Клас FileManager відповідальний за роботу з файлами в програмному засобі Visu і має такі методи:

а) open_file() – відповідає за відкриття конфігураційного файлу у форматі .csv;

б) upload_file() – відповідає за завантаження конфігураційного файлу до програмного засобу;

в) save_file() – відповідає за збереження файлу у програмному засобі;

г) parse() – відповідає за парсинг даних, прочитаних з конфігураційного файлу та перетворення цих даних у відповідні запити до бази даних.

Клас ConfigurationWindow – клас графічного інтерфейсу, що показує користувачу конфігураційне вікно, на якому є текстовий надпис, що треба завантажити файл у форматі csv та сама кнопка завантаження. В даному класі також реалізований фільтр, який дозволяє завантажувати до системи тільки файли у зазначеному форматі. Також після завантаження файлу до системи, виникає вікно з повідомленням про результат операції. Даний клас має тільки два методи:

а) upload_file() – відповідає за відкриття файлового провідника на комп’ютері користувача та фільтрацію файлів, що вибираються користувачем у цьому провіднику;

б) show_result_window() – відповідає за результат завантаження конфігураційного файлу користувачем.

У пакеті ui знаходяться усі програмні класи графічного інтерфейсу. Даний пакет розділений ще на два: widgets та windows відповідно – для віджетів і для вікон, які розміщують в собі відповідні віджети. У пакеті widgets знаходяться програмні класи, відповідальні за відображення віджетів для кожної з чотирьох діаграм (WidgetDisciplineToLearningOutcome, WidgetAmountDiscToLearningOutcome, WidgetSemToLearningOutcome, WidgetSemToAmountLearningOutcome). У цьому випадку віджети виступають як контейнери, у якому знаходяться канви, на яких і малюються діаграми. Більш детально пакет ui.widgets можна розглянути на рис. 4.8 (виділені зеленим кольором).

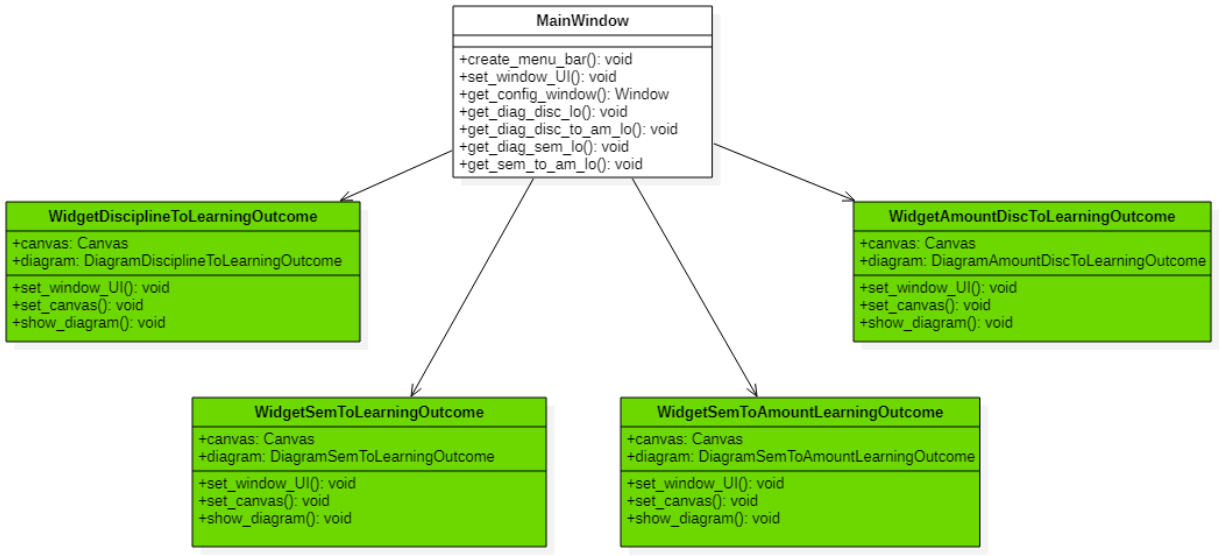

Рисунок 4.8 – Структура пакету ui.widgets

Всі зображені на рис. 4.8 класи-віджети мають аналогічну структуру: в змінних canvas зберігаються об’єкти класу QCanvas, що являються полотнами для візуалізації самих діаграм, в змінних diagram – знаходяться ті типи діаграм, які вибрав користувач у певний момент часу, set_window_UI() метод задає загальні властивості розміру вікна та елементів, що в ньому знаходяться, set_canvas() – встановлює для об’єкта віджета об’єкт канви, show_diagram() – метод, що показує користувачу на екрані той тип діаграми, яку він обрав.

Як було зазначено раніше, метод set_window_UI() задає властивості графічних елементів таких як:

а) розмір вікна;

б) шрифт, розмір і колір тексту;

в) точні координати розташування графічного елементу на віджеті;

г) колір і розмір самого графічного елементу;

ґ) анімація тощо.

В пакеті diagrams знаходяться ті програмні класи, що відповідальні за чотири діаграми візуалізації освітньої програми, які доступні користувачу. Клас DiagramDisciplineToLearningOutcome предоставляє діаграму, на якій зображені дисципліни та результати навчання, які наявні у цих дисциплін,

DiagramAmountDiscToLearningOutcome – діаграму, на якій зображено відношення дисциплін до певних кількостей результатів навчання, DiagramSemToLearningOutcome – діаграму, на якій зображено всі семестри та результати навчання, які проводилися в кожному з них, DiagramSemToAmountLearningOutcome – діаграму, що показує всі семестри та кількості результатів навчання, які були в кожному з них.

Як висновок, у даному розділі був проведений аналіз існуючих архітектур програмного забезпечення, в якому виявилися усі переваги та недоліки кожної з них. Найкращим варіантом для програмного засобу Visu стала багатошарова архітектура, що складається з трьох шарів: шару Представлення, шару Бізнес-логіки та шару Передачі даних. Були обрані технології розробки для програмного засобу, а саме: мова програмування Python, бібліотеки для графічного інтерфейсу PyQt, середовище розробки для написання та відладки коду. Також були наведені діаграми послідовності для п’яти варіантів використання:

а) завантаження файлу до системи Visu;

б) відображення діаграми дисциплін та їх результатів навчання;

в) відображення діаграми кількості результатів навчання до дисциплін;

г) відображення діаграми семестрів та результатів навчання;

ґ) відображення діаграми семестрів та кількості результатів навчання.

Окрім цього, у цьому розділі була наведена діаграма програмних класів програмного засобу Visu та структура пакету ui.widgets та детальний опис класів, що в ньому знаходяться. Для діаграми програмних класів також було надано детальний опис ключових програмних класів, їх методів та призначення цих методів.

5 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

Після того, як вимоги та дизайн продукту затверджені, відбувається перехід до наступної стадії життєвого циклу – безпосередньо розробки. Тут починається написання програмістами коду програми відповідно до раніше визначених вимог. На цьому етапі налаштовується програмне оточення, розробляється інтерфейс програми, пишуться програмні класи, що реалізують бізнес-логіку, а такж пишеться логіка взаємодії програмної системи з базою даних. Крім того, програмісти пишуть Unit-тести для перевірки правильності роботи коду кожного компонента системи, проводять рев'ю написаного коду, створюють програмні артефакти та розгортають готове програмне забезпечення у програмному середовищі. Цей цикл повторюється доти, доки всі вимоги не будуть реалізовані. Зазвичай програмна реалізація складається з чотирьох стадій: розробки алгоритмів, написання коду, компіляції, тестування та відладки.

5.1 Організація даних

Оскільки на етапі проектування було вирішено використати реляційну базу даних та систему управління базою даних PostgreSQL, то дані, що описують властивості освітньої програми, а саме дисципліни, їх результати навчання та семестри, в яких викладалися зазначені дисципліни, завантажені користувачем до програмної системи у спеціальному форматі, повинні зберігатися в реляційних таблицях бази даних. У програмній системі можна виділити три основні сутності, що будуть містити всі дані. Між цими сутностями існують певні зв’язки, які надають можливість з’єднувати їх і таким чином отримати максимально повну інформацію щодо кожної сутності. Сутність Освітня Компонента (Discipline) і Результат Навчання (LearningOutcome) зв’язані між собою іменованою асоціацію «має». Також вони з’єднані між собою відношенням «багато до багатьох», тому що одна дисципліна має багато результатів

навчання, а один й той самий результат навчання може бути у декількох дисциплінах одночасно. Загалом всі сутності і зв’язки між ними можна представити у вигляді концептуальної моделі даних (ER-моделі (англ. - Entity Relationship)) (рис. 5.1):

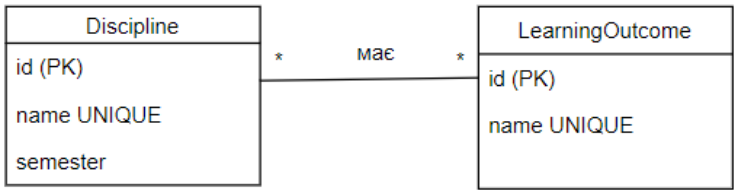

Рисунок 5.1 - ER-модель бази даних програмного засобу Visu

У подальшому створену концептуальну модель слід представити у реляційному вигляді (рис. 5.2):

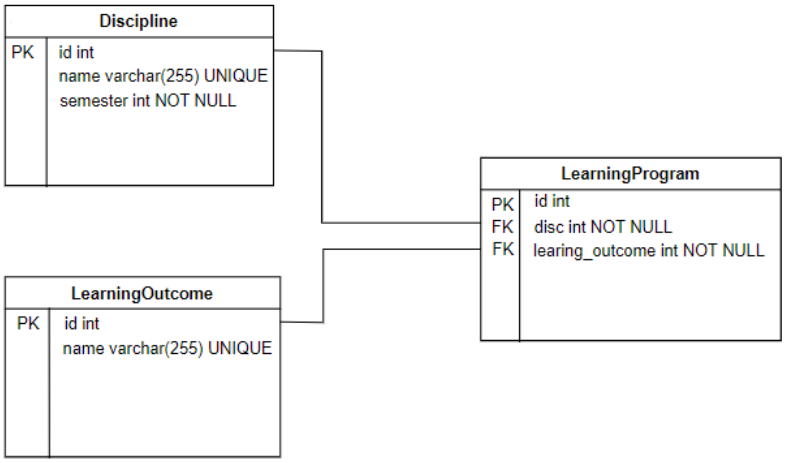

Рисунок 5.2 – Реляційний вигляд бази даних програмного засобу Visu

Реляційний вигляд означає, що сутності представлені у вигляді двомірних таблиць, дотримуючись усіх правил перетворення концептуальної моделі даних у реляційну модель, а саме:

а) якщо між таблицями А і Б існує зв’язок 1:1 (один до одного), то в таблиці Б повинен бути вторинний ключ, що відповідає первинному з таблиці А;

б) якщо між таблицями А і Б існує зв’язок 1:m (один до багатьох), то в таблиці Б має бути вторинний ключ, що буде вказувати на єдиний варіант з таблиці А;

в) якщо між таблицями А і Б існує зв’язок n:m (багато до багатьох), то слід створити окрему таблицю, в якій будуть два вторинних ключа, кожен з яких буде відповідати первинному ключу відповідних таблиць.

Оскільки таблиці Discipline та LearningOutcome пов’язані відношенням «багато до багатьох», то з’явилася третя допоміжна таблиця LearningProgram, що описує властивості освітньої програми.

Наведемо детальний опис кожної з таблиць.

Таблиця 5.1 - Опис таблиці Discipline

Назва	Тип	Ключ	Пояснення
id	int (serial)	primary key	первинний ключ таблиці
name	varchar(255)	-	назва дисципліни
semester	int	-	семестр

Приклад створення таблиці:

```
create table if not exists Discipline(
id serial primary key,
name varchar(255) unique not null,
semester int not null,
check (semester >0 and semester <9)
);
```

Слід зазначити, що семестр повинен бути від 1 до 8.

Таблиця 5.2 – Опис таблиці LearningOutcome

Назва	Тип	Ключ	Пояснення
id	int (serial)	primary key	первинний ключ таблиці
name	varchar(255)	-	назва навчального результату

Приклад створення таблиці:

```
create table if not exists LearningOutcome (
id serial primary key,
name varchar(255) unique not null
);
```

Сутності Discipline та LearningOutcome зв’язані між собою відношенням many-to-many, отже ми повинні створити додаткову таблицю (табл. 5.3), в якій будуть зберігатися значення і з першої, і з другої таблиці.

Таблиця 5.3 – Опис таблиці LearningProgram

Назва	Тип	Ключ	Пояснення
id	int (serial)	primary key	первинний ключ таблиці
disc	int	foreign key	вторинний ключ таблиці, що зв’язує з таблицею Discipline
learning_outcome	int	foreign key	вторинний ключ таблиці, що зв’язує з таблицею LearningOutcome

Приклад створення таблиці:

```
create table if not exists LearningProgram(
id serial primary key,
disc integer not null,
learning_outcome integer not null,
constraint fk_disc foreign key(disc) references Discipline(id),
constraint fk_lo foreign key(learning_outcome) references LearningOutcome(id)
);
```

У цьому підрозділі була описана база даних, яка зберігає дані, завантажені користувачем з файлу, що описують властивості освітньої програми, а саме дві її основні сутності – Дисципліна та Результат Навчання.

5.2 Структура конфігураційного файлу користувача

Як було зазначено у функціональних вимогах, програмний засіб Visu надає можливість користувачу завантажити файл з власними даними до програмного засобу. Завантажити файл користувач може через конфігураційне вікно Visu і розширення файлу може бути тільки .csv. У конфігураційному файлі знаходяться властивості освітньої програми, а саме: дисципліни, результати навчання, семестри. Заповнювати цей файл потрібно за певним шаблоном, показаним на рисунку 5.3:

	A	B	C
1	Discipline	Semester	LearningOutcome

Рисунок 5.3 – Шаблон для конфігураційного файлу користувача

Кожний рядок повинен містити умовне позначення дисципліни (наприклад D1, D2 і т.д), семестр – число від одного до восьми, та умовне позначення результату

навчання (наприклад РН1, РН2 і т.д). Приклад правильного заповнення конфігураційного файлу зображено на рисунку 5.4:

	A	B	C
1	Discipline	Semester	LearningOutcome
2	D1	1	PH1
3	D1	1	PH2
4	D2	1	PH4

Рисунок 5.4 – Приклад заповнення конфігураційного файлу

Слід зазначити, що файл з будь-якому іншому форматі користувач не зможе вибрати через файловий провідник, щоб потім завантажити до Visu.

5.4 Алгоритми для реалізації функцій програмного засобу

Для програмної реалізації будь-якої програмної системи використовують алгоритми. Алгоритм – це скінчена послідовність команд (вказівок), що визначає, які дії та у якому порядку потрібно виконати, щоб досягти поставленої мети [10]. Зазвичай більшу частину алгоритмів не потрібно писати самостійно, вони вже реалізовані в спеціальних бібліотеках, що ретельно тестуються і перевіряються на максимальну ефективність. Проте існує функціонал, який потрібно писати самостійно, і в такому разі складання алгоритмів є не тільки ще одним видом документації програмної системи, а й допомогою при написанні коду. Представимо на рисунку 5.5 алгоритм для реалізації функції завантаження користувачем файлу з даними до системи.

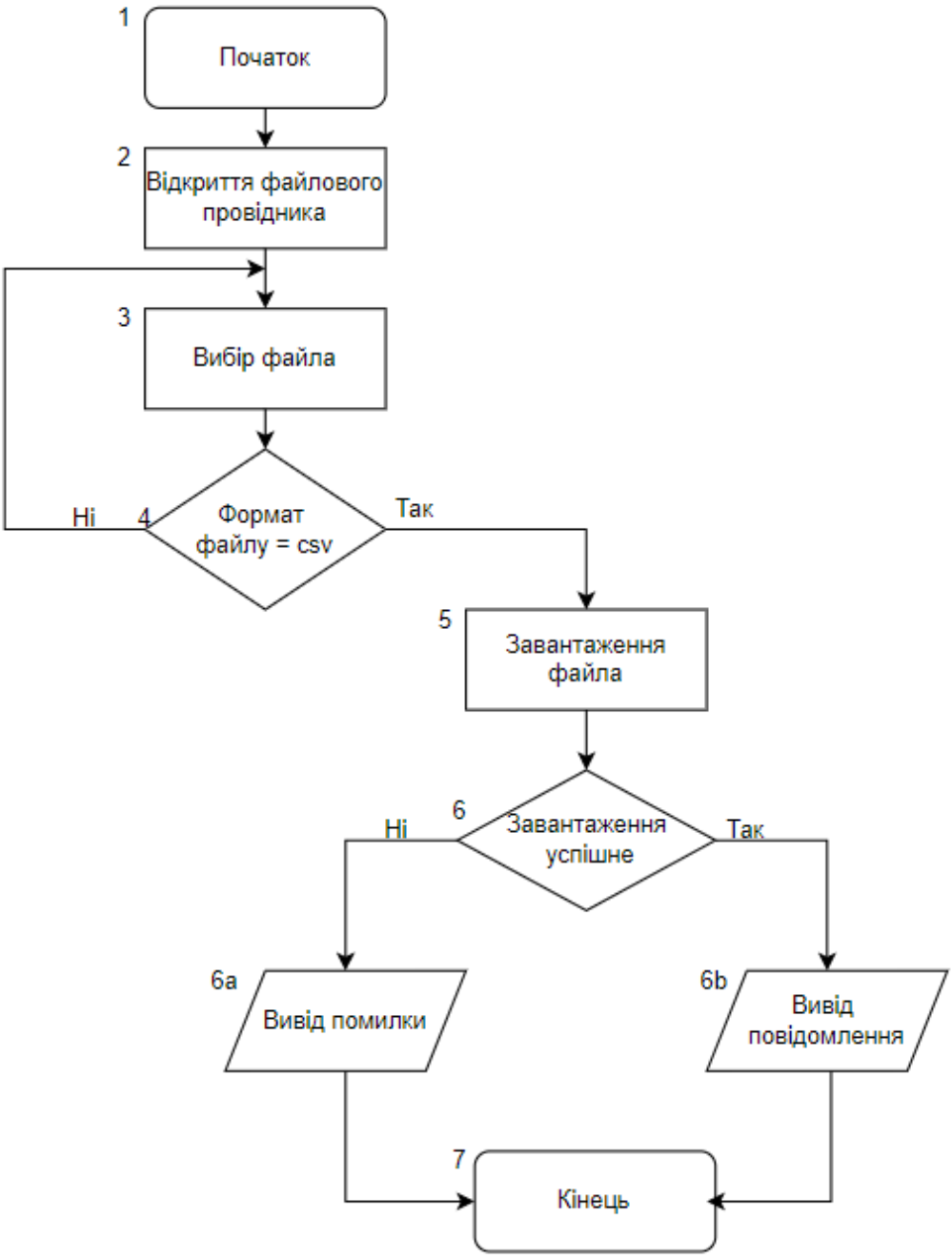

Рисунок 5.5 – Алгоритм для реалізації функції завантаження файлу користувачем до системи Visu

Пояснення алгоритму:

1. Початок алгоритму.
2. Відкриття файлового провідника на комп’ютері користувача.

3. Вибір файла до файлового провідника.

4. Перевірка формату файлу, який вибрав користувач. Якщо файл має формат csv, то перехід до п.5. Інакше перехід до п.3.

5. Завантаження файлу до програмного засобу Visu.

6. Перевірка результату завантаження. Якщо завантаження файлу успішне, то перехід до п. 6b, інакше – перехід до п. 6a.

6a. Вивід на екран помилки.

6b. Вивід повідомлення про успішне завантаження файлу.

7. Кінець алгоритму.

Однією з найбільш значущих властивостей алгоритму вважається його складність. Зазвичай складність оцінюють за кількістю дій, які виконує алгоритм, та за обсягом задіяної пам’яті. Кількість вхідних даних прийнято позначати літерою n. Для позначення оцінки складності алгоритмів використовують так звану O-нотацію. У даному алгоритмі на вхід подається тільки 1 файл, отже складність алгоритму складає O (1).

На рисунку 5.6 представимо алгоритм для реалізації функції отримання даних для діаграми з бази даних за допомогою мови SQL. Цей алгоритм підходить для усіх діаграм, і відрізняється тільки запитом, який використовуються для отримання даних для кожної діаграми окремо.

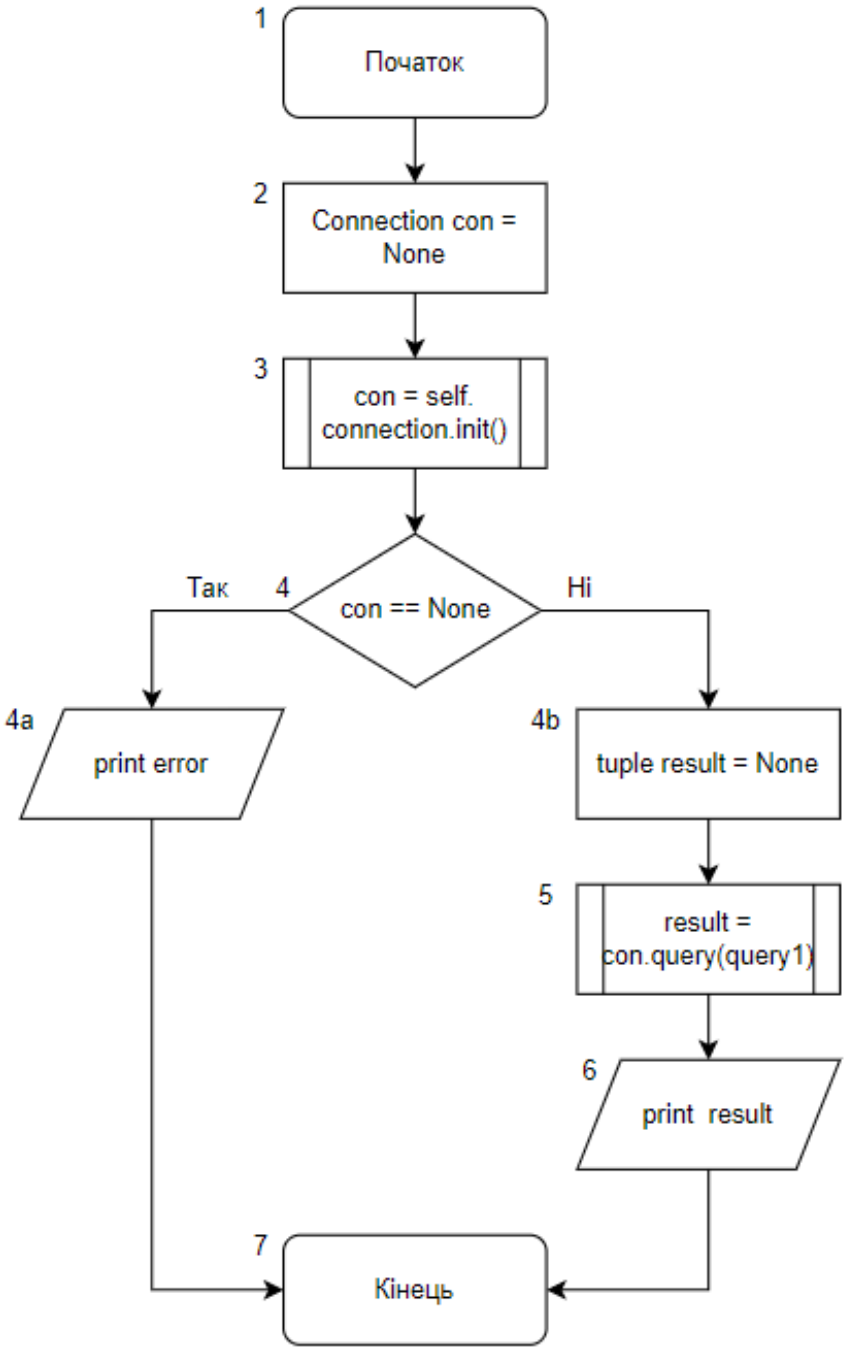

Рисунок 5.6 – Алгоритм для реалізації функції отримання даних для діаграми з бази даних

Пояснення алгоритму:

1. Початок алгоритму.

2. Створення змінної з назвою con і типом Connection.

3. Запис результату метода connection.init() в змінну con. Метод connection.init() Створює новий зв’язок з базою даних.

4. Перевірка на None змінної con. Якщо зв’язок з базою не зміг встановитися, то в змінну con не запишеться нічого – перехід до п.4a і виведеться на екран користувачу помилка, інакше перехід до п. 4b.

a. Виведення на екран користувачу тексту помилки.

b. Створення змінної з назвою result та типом Tuple.

5. Запис в змінну result результату виконання методу con.query(query1) (метод викликає у об’єкта con запит на виконання SQL-запиту, що записаний в query1).

6. Вивід на екран результату.

7. Кінець алгоритму.

Складність цього алгоритму залежить від запиту query1, оскільки якщо це звичайна вибірка даних, то це складність O(n). В іншому випадку може бути підвибірка вибраних даних чи сортування, і складність алгоритму зміниться.

5.4 Реалізація інтерфейсу користувача

Інтерфейс користувача – це зовнішній вигляд програмної системи, за допомогою якого користувач має можливість використовувати функціонал системи. Слід зазначити, що велику роль відіграє зрозумілість інтерфейсу, адже таким чином користувач не буде витрачати багато часу на пошуки кнопок чи інших графічних елементів, щоб виконати бажані дії.

До інтерфейсу користувача існують такі вимоги як:

1. Ясність – інтерфейс дозволяє уникати двозначності, тобто назва чи вид елементу повинен відповідати його цілі. Кнопка з назвою «Завантажити зображення» повинна завантажувати тільки зображення, а не файли чи аудіо.

2. Виразність – інтерфейс повинен бути прозорим, не слід використовувати надмірні уточнення, спливаючі підказки для абсолютно всіх елементів, адже

призводить до «роздування» інтерфейсу. Окрім того, надмірна кількість графічних елементів заважає знайти в першу чергу потрібні елементи. Треба дотримуватися правила, що інтерфейс користувача повинен бути одночасно стислим і зрозумілим.

3. Знайомість – слід використовувати загальні позначки елементів, які вже використовуються в інших програмних системах, щоб користувач впізнав їх, навіть якщо бачить інтерфейс нової системи вперше.

4. Відповідність – відмінний інтерфейс мусить забезпечувати користувача хорошим зворотним зв'язком відносно того, що відбувається, наприклад, пункти, які повинен вибрати користувач, повинні виділятися кольором, шрифтом або розміром, щоб виділятися серед інших пунктів.

5. Послідовність – важливо, щоб інтерфейс і функціонал програмної системи були узгоджені, тобто дані, які будуть використовуватися для певних цілей програмою, повинні вводитися користувачем на самому початку.

6. Естетика – інтерфейс, окрім надання можливості користуватися функціоналом програмної системи, повинен мати приємний вигляд. Слід уникати великої кількості кольорів, використання різноманітних шрифтів тощо.

Користувач відкриває програмний засіб Visu і бачить вікно з чотирма кнопками: «Діаграма дисциплін та їх результатів навчання», «Діаграма кількості результатів навчання до дисциплін», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів до кількості результатів навчання» відповідно. Також у верхньому лівому кутку знаходиться меню з двома опціями «Діаграми» та «Конфігурація». На рис. 5.4 показано зображення цього вікна.

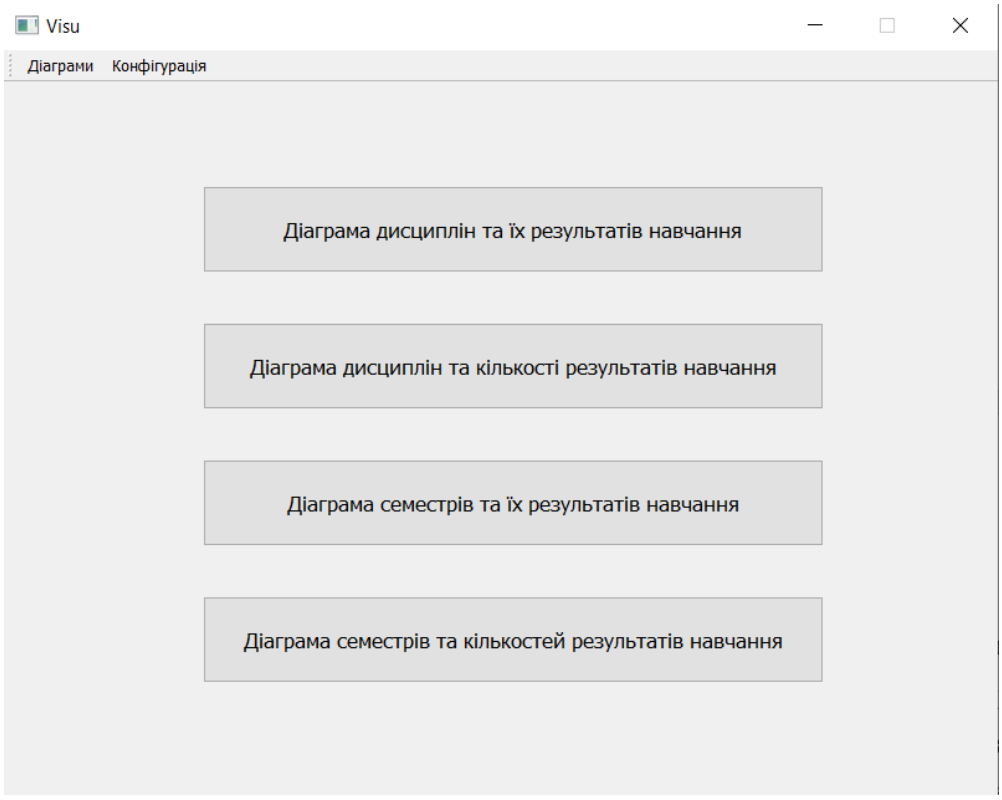

Рисунок 5.4 – Зображення головного вікна Visu

Вкладка «Діаграми» також відображає користувачу головне вікно. Для того, щоб завантажити файл до програмної системи, користувачу слід перейти на вкладку «Конфігурація», після чого з’явиться вікно конфігурації з надписом та кнопкою завантаження (рис. 5.5). Після натиснення кнопки «Завантажити», програмна система відкриває файловий провідник на комп’ютері користувача, в якому він зможе обрати потрібний йому файл з даними. Слід зазначити, що у файловому провіднику можна тільки файли з розширенням .csv будуть доступні.

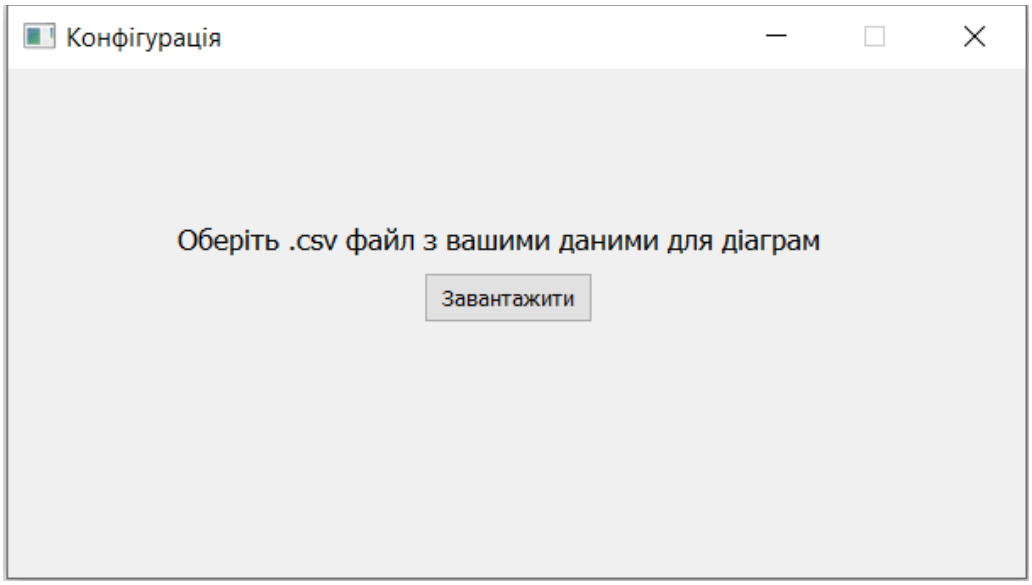

Рисунок 5.5 – Зображення конфігураційного вікна Visu

Після успішного завантаження файлу повинне з’явитися інформаційне вікно з повідомленням, що завантаження пройшло успішно (рис. 5.6):

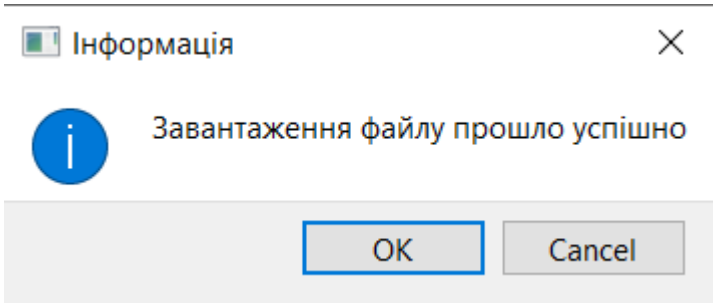

Рисунок 5.6 – Зображення інформаційного вікна Visu

На вкладці «Діаграми», якщо користувач натисне кнопку «Діаграма дисциплін та їх результатів навчання», то з’явиться нове вікно з гістограмою, де по осі абсцис знаходяться усі можливі результати навчання, а по осі ординат – кількість дисциплін, які містять вказані результати навчання (рис. 5.7). Слід зазначити, що для візуалізації даних, що відображають кількість, найкращим рішенням є відображення цих даних у вигляді гістограми - стовпчастої діаграми, адже таким чином можна побачити відразу,

яких результатів навчання найбільше, порівняно з іншими, і яких результатів навчання найменше.

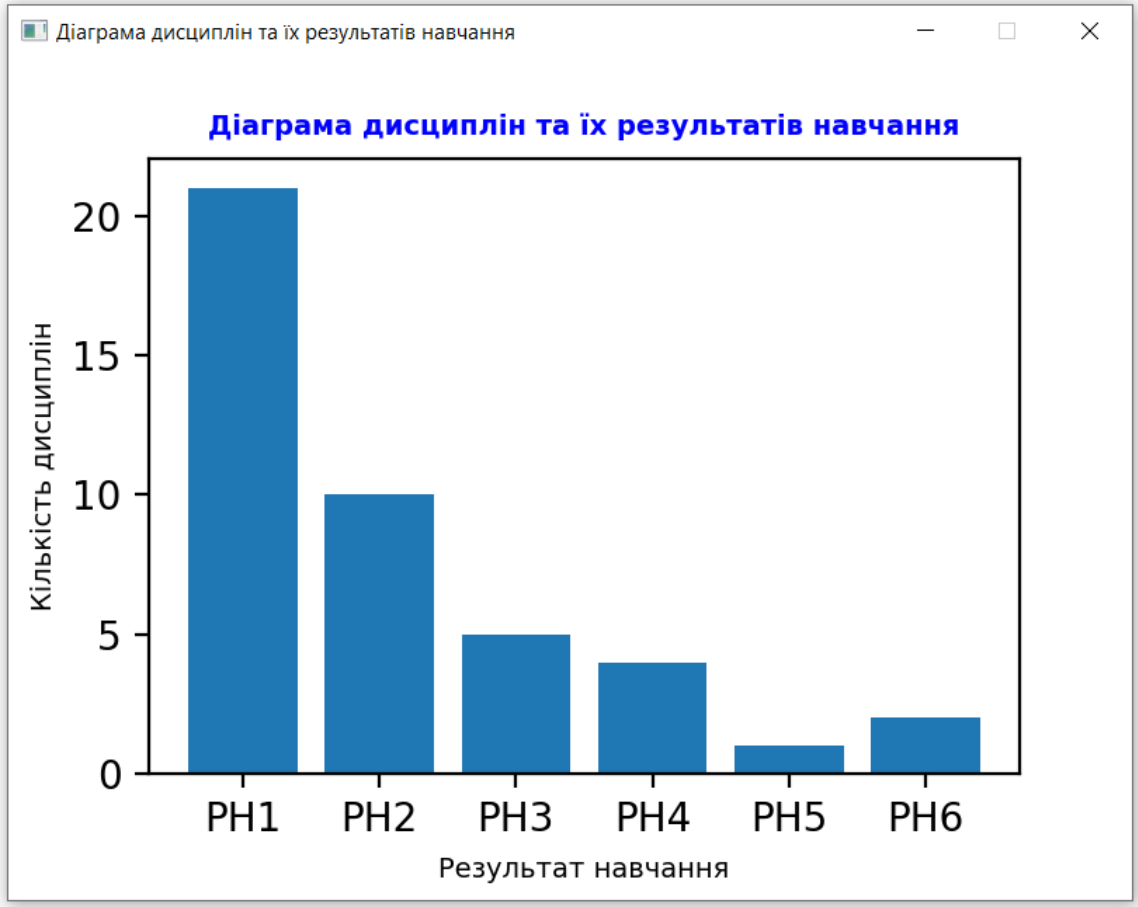

Рисунок 5.7 – Діаграма дисциплін та їх результатів навчання

Якщо користувач натисне кнопку «Діаграма кількості результатів навчання до дисциплін», то система відкриє нове вікно з зображенням гістограми, де по осі абсцис знаходиться кількість результатів навчання, а по осі ординат – кількість дисциплін, які містять вказану кількість результатів навчання (рис.5.8). Наприклад, відбираються спочатку всі дисципліни та кількість результатів навчання, що були у цієї дисципліни на протязі навчального року. Потім з цієї вибірки відбираються пари – одна кількість навчання зустрічається в n-кількості дисциплінах, друга кількість результатів навчання – в m-ній кількості дисциплін тощо.

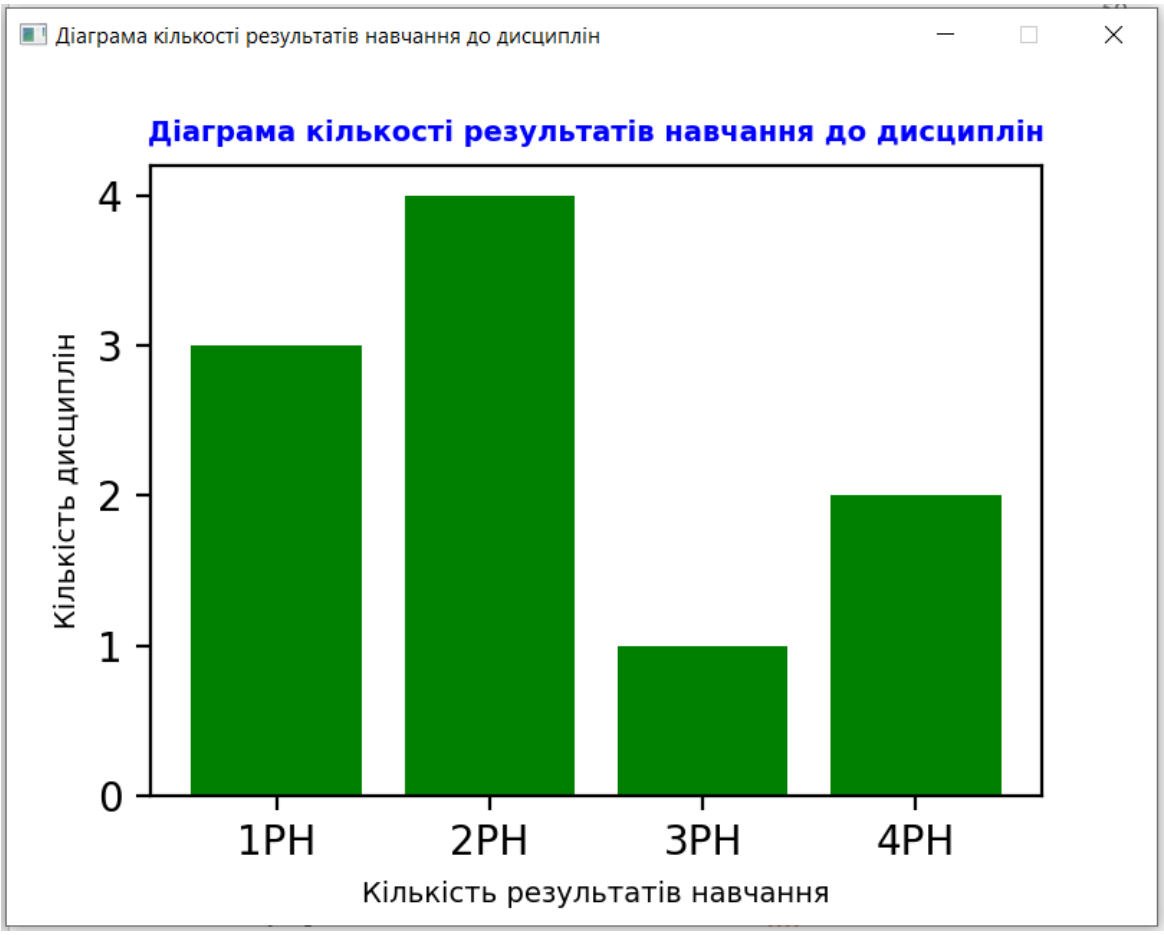

Рисунок 5.8 – Діаграма кількості результатів навчання до дисциплін

На рис. 5.8 можна побачити, що по одному результаті навчання зустрічаються в трьох дисциплінах, по два результати навчання – в чотирьох дисциплінах, три результати навчання є тільки в одній дисципліні, а по чотири результати навчання є тільки в двох дисциплінах тощо.

Коли користувач натискає на кнопку «Діаграма семестрів та їх результатів навчання», то система відкриває нове вікно з горизонтальною діаграмою, що зображена на рис. 5.9. Саме на горизонтальній діаграмі можна легко побачити наявність або відсутність конкретного результату навчання для конкретного семестру. Для зручності кожен семестр має власний колір для тих результатів навчання, які в ньому знаходилися. Лінійний графік, наприклад, створений на основі цих даних, мав

би вигляд просто крапок, розташованих за певними координатами, і був би незручним для аналізу цього графіку.

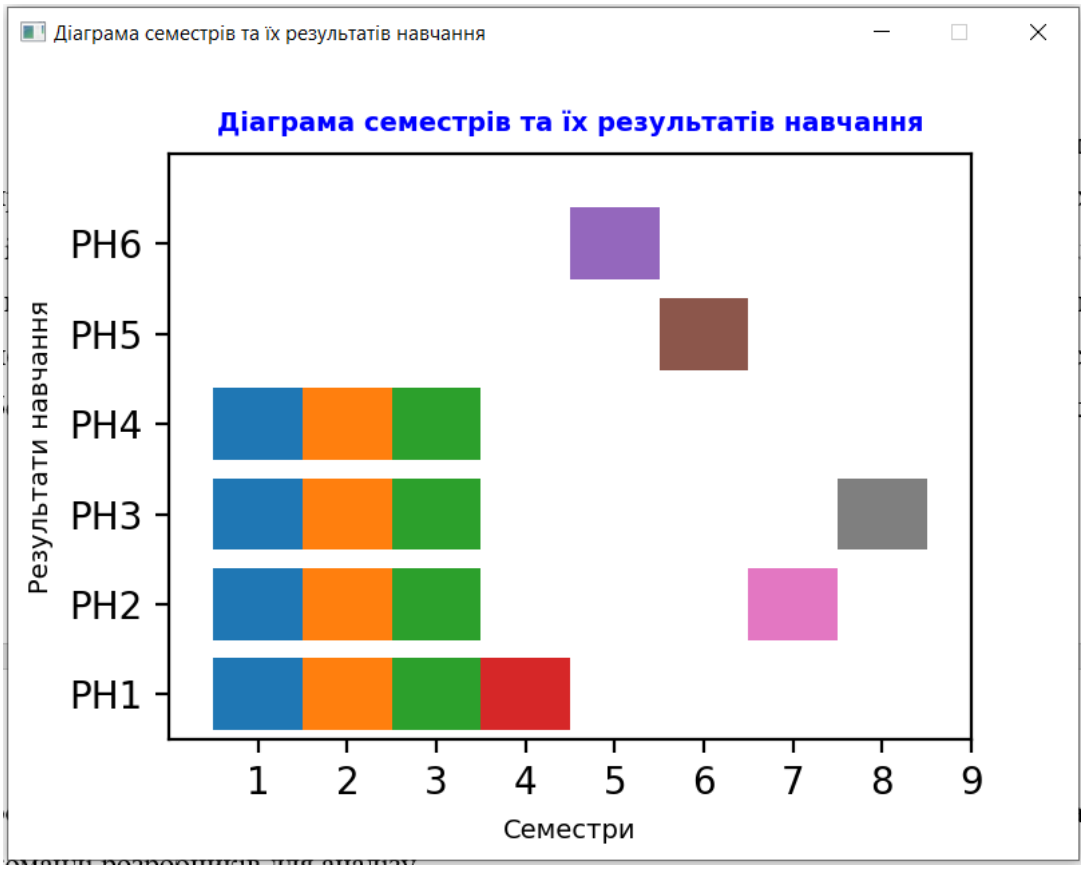

Рисунок 5.9 – Діаграма семестрів та їх результатів навчання

На даній діаграмі показується, які саме результати навчання були в кожному семестрі. По осі абсцис зображені семестри, по осі ординат – результати навчання, і видно, що в першу семестрі проводилися РН1, РН2, РН3, РН4, в сьомому семестрі – тільки РН2, в п'ятому – тільки РН6.

На рис. 5.10 зображена діаграма «Діаграма семестрів до кількості результатів навчання». Вона з’являється, якщо користувач натисне відповідну кнопку. Діаграма має вигляд павука, яка показує, скільки результатів навчання проведено в кожному семестрі.

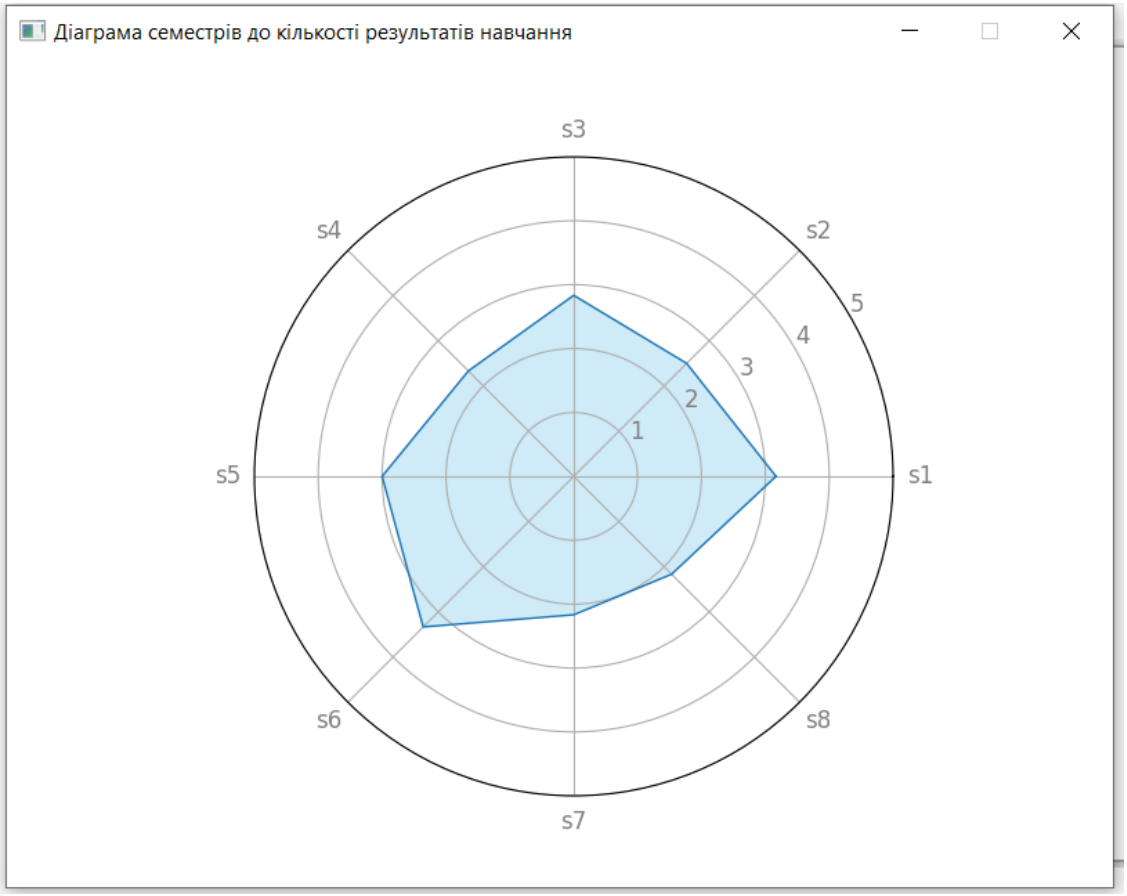

Рисунок 5.10 - Діаграма семестрів до кількості результатів навчання

Згідно з зображенням діаграми, в першому семестрі (s1) та шостому (s6) було найбільше результатів навчання.

У даному розділі був описаний функціонал, який реалізує вимоги до програмного засобу Visu, описана структура бази даних, наведені два алгоритми для двох варіантів використання «Завантаження файлу користувачем до системи» та «Отримання даних для діаграми з бази даних» у вигляді блок-схем та словесним поясненням. Також були наведені приклади графічного інтерфейсу користувача, а саме показані зображення усіх чотирьох діаграм, головного вікна програмного засобу, конфігураційного вікна для завантаження користувачем файлу з даними до системи, інформаційного вікна з повідомленням про результат завантаження файлу.

6 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСОБУ

Після програмної реалізації системи треба провести тестування, щоб перевірити правильність результатів виконання усіх функцій системи. Тестування програмного забезпечення – це сукупність процесів, серед яких перевірка відповідності розробленої системи вимогам, які заздалегідь були сформовані; виявлення технічних помилок (багів), відповідність розробленої системи нефункціональним вимогам (оцінка зручності використання, продуктивності, безпеки, локалізації) [12].

Існують багато видів тестування, кожен з яких направлений на переслідувану ціль. Наприклад, в залежності від знання програмної системи, тестування може бути трьох видів: тестування чорної скриньки (black box), тестування білої скриньки (white box), тестування сірої скриньки (gray box). Black box – тестування, засноване на специфікації, для роботи виключно із зовнішніми інтерфейсами системи. White box передбачає, що внутрішня структура системи відомі тестувальнику, і він вибирає вхідні значення, ґрунтуючись на знанні коду, який їх оброблятиме. Окрім цього, він знає, яким має бути результат цієї обробки. Тестування білого ящика – поглиблення у внутрішній пристрій системи, поза її зовнішніх інтерфейсів. Gray box – це комбінація тестування чорного та білого ящиків.

Також тестування буває функціональним, нефункціональним та регресійним. Функціональне тестування означає, що воно направлене на відповідність функціоналу програмної системи бізнес-вимогам, а нефункціональне – оцінює такі аспекти системи як продуктивність, зручність використання, безпека тощо. Регресійне тестування у свою чергу перевіряє вже протестований функціонал після того, як до функціоналу внесли певні зміни. Таке тестування проводиться, щоб переконатися, що програмна система працює як повинна з новим функціоналом, з виправленими помилками [11].

Тестування є дуже важливим етапом при створенні будь-якої програмної системи, тому що виявлені помилки при роботі системи на фінальних стадіях розробки набагато дорожчі, ніж їх виявлення на самому початку.

6.1 Тестові випадки для програмного засобу

Невід'ємною частиною будь-якого тестування є test-case (тестовий випадок). Його мета – виконати певну послідовність дій, необхідних для перевірки конкретної функціональності. Він обов'язково складається з кількох частин: ідентифікатора, назви, дії, очікуваних результатів.

В табл. 6.1 наведемо тестовий випадок для одного з варіантів використання (рис. 3.1) «Завантаження файлу з даними».

Таблиця 6.1 – Тестовий випадок для завантаження файлу з даними:

Назва характеристики	Значення характеристики
Ідентифікатор	TC1
Назва	Дозволяє перевірити реакцію програмної системи на завантаження файлу користувачем.
Передумова	Користувач відкрив програмний засіб Visu.
Опис дій	1. Користувач перейшов на вкладку «Конфігурація». 2. Користувач натиснув кнопку «Завантажити». 3. Користувач обрав файл та натиснув «Відкрити».
Очікуваний результат	1. На екрані з'явилося додаткове вікно з назвою «Конфігурація», надписом «Оберіть .csv файл з вашими даними для діаграм», кнопкою «Завантажити». 2. На екрані з'явилося вікно файлового провідника комп'ютера. 3. Файл завантажився до системи успішно. На екрані з'явилося вікно з повідомленням «Завантаження файлу пройшло успішно».

В табл. 6.2 наведемо тестовий випадок для варіанту використання «Перегляд діаграми дисциплін та їх результатів навчання»:

Таблиця 6.2 – Тестовий випадок для перегляду діаграми дисциплін та їх результатів навчання

Назва характеристики	Значення характеристики
Ідентифікатор	ТС2
Назва	Дозволяє перевірити зображення діаграми для дисциплін та їх результатів навчання.
Передумова	Користувач відкрив програмний засіб Visu.
Опис дій	1. Користувач перейшов на вкладку «Діаграми». 2. Користувач натиснув кнопку «Діаграма дисциплін та їх результатів навчання».
Очікуваний результат	1. На екрані з'явилося вікно з чотирма кнопками: «Діаграма дисциплін та їх результатів навчання», «Діаграма дисциплін та кількості результатів навчання», «Діаграма семестрів та їх результатів навчання», «Діаграма семестрів та кількостей результатів навчання». 2. На екрані з’явилося нове вікно, на якому намальована діаграма, по осі абсцис розташовані результати навчання, по осі ординат – кількість дисциплін.

Всього було складено десять тестових випадків - п’ять випадків для опису варіантів використання програмного засобу Visu, чотири – для опису правильного зображення візуалізованих діаграм (еквівалентність очікуваного та фактичного зображення діаграм), один – для перевірки правильності даних, які знаходяться в

файлі, що завантажує користувач до програмного засобу. Для останнього тест-кейсу мається на увазі опис реакції програмного засобу, який не зможе завантажити дані користувача з файлу через помилку парсингу – наприклад відсутність крапки з комою між рядками. У даному підрозділі були описані тільки два тестових випадка для двох варіантів використання: «Завантаження файлу з даними», «Перегляд діаграми дисциплін та їх результатів навчання». Тестування по першим п’яти тест-кейсам проводилося «чорним ящиком», тобто перевірялася виключно взаємодія користувача з інтерфейсом системи. Усі п’ять тестів були пройдені успішно. Також було виконане тестування «білим ящиком» діаграм програмного засобу, де на вхід подавалися дані, необхідні для побудови діаграм, а на виході перевірялося правильне відображення (візуалізація) кожної з діаграм. Тестування «білим ящиком» також було пройдено успішно. Не було виконане тестування по останньому тест-кейсу – перевірці правильності даних, завантажених файлом до системи - через недостачу часу. Як висновок, тестування було проведено по дев’яти тест-кейсам із десяти, тобто покриття тестами програмного засобу Visu склало 90%.

6.2 Визначення ефективності програмного засобу

Ефективність програмної системи в загальному розумінні – це здатність програмної системи забезпечити відповідну продуктивність в залежності від кількості використовуваних ресурсів при заданих умовах. Під ефективністю програмного засобу Visu мається на увазі досягнення мети, зазначеної на самому початку кваліфікаційної роботи - зменшення кількості годин аналізу властивостей освітньої програми, які відображають залежність між дисциплінами та результатами навчання за певний період часу, за допомогою візуалізації даних у вигляді діаграм. Для того щоб підтвердити, що мета була досягнута, був проведений експеримент, в якому порівнювався час, затрачений на аналіз освітньої програми ручним способом

та за допомогою програмного засобу Visu. Експеримент був проведений у декілька етапів.

1 етап – підготовка. Було вибрано п’ять осіб, що мають досвід в складанні або оцінюванні освітніх програм. Особам була видана одна освітня програма, яку потрібно було проаналізувати два рази: без використання програмного засобу Visu – перше завдання, а потім з її використанням – друге завдання. Освітню програму потрібно було проаналізувати за чотирма критеріями: порахувати які результати навчання були в кожній дисципліні кожного семестру в одному навчальному році, скільки результатів навчання було в кожній дисципліні в кожному семестрі, яких результатів навчання з усіх семестрів зустрічалося найчастіше, які результати навчання взагалі були в кожному семестрі.

2 етап – постановка задачі. Кожна з п’яти осіб повинна була спочатку самостійно, не використовуючи жодні програмні засоби, зробити аналіз властивостей виданої освітньої програми за зазначеними чотирма критеріями. Потім цим же особам було видано друге завдання - внести в файл в форматі csv дані про властивості виданої освітньої програми: дисципліни, в якому семестрі дисципліни проводилися, результати навчання кожної дисципліни за шаблоном. Даний файл потрібно було завантажити до програмного засобу і також виконати аналіз за зазначеними вище критеріями.

3 етап – виконання. Кількість витрачених годин на виконання кожною особою кожного завдання записувалася.

4 етап – порівняння. Результати експерименту наведено в таблиці 6.3.

Особа 1 витратила на самостійний аналіз освітньої програми п’ять годин, а за допомогою програмного засобу Visu – три години. Друга особа – Особа 2 – витратила на самостійний аналіз три години, а за допомогою Visu – тільки дві години. В цілому, як видно з таблиці 6.3, самостійний аналіз для кожної особи, що брала участь в експерименті, тривав довше, ніж при використанні розробленого програмного засобу.

Таблиця 6.3 – Результати експерименту

Особа	Самостійний аналіз властивостей освітньої програми, год	Аналіз властивостей освітньої програми за допомогою Visu, год
Особа 1	5	3
Особа 2	3	2
Особа 3	5	4
Особа 4	3,7	2,9
Особа 5	4,5	4

Далі були розраховані середня кількість годин, витрачених кожної особою на виконання кожного з двох завдань (самостійного аналізу властивостей освітньої програми та їх аналізу за допомогою програмного засобу Visu) за формулою:

$$T_c = \frac{1}{n}\sum_{i=1}^{n} T_i \,, \tag{6.1}$$

де *Tc* – середня кількість годин, витрачених особами на виконання аналізу освітньої програми, *n* – кількість осіб, що брали участь в експерименті, *Ti* – кількість годин аналізу освітньої програми кожної особи.

Підставивши у формулу 6.1 дані з таблиці 6.3 порахуємо скільки в середньому годин було витрачено на виконання першого завдання і скільки – другого завдання.

На виконання першого завдання, тобто аналізу властивостей освітньої програми, без використання програмних засобів було витрачено в середньому (Tc_1) годин для *n*=5 осіб:

$$Tc_1 = \frac{5 + 3 + 5 + 3.7 + 4.5}{5}$$

Звідси Tc_1 = 4.25 годин.

Порахуємо середню кількість годин на виконання п’ятьма особами другого завдання, тобто аналізу властивостей освітньої програми з використанням програмного засобу Visu (Tc_2):

$$Tc_2 = \frac{3 + 2 + 4 + 2.9 + 4}{5}$$

Звідси Tc_2 = 3.18 годин.

По формулі 6.2 обчислимо різницю між середніми кількостями годин виконання двох завдань:

$$Tc = Tc_1 - Tc_2 \qquad (6.2)$$

Підставивши замість Tc_1 = 4.25, а замість Tc_2 = 3.18, отримаємо 4.25 – 3.18 = 1.07.

Отже, можемо зробити висновок, що аналіз властивостей освітньої програми за допомогою програмного засобу Visu на 1.07 годину швидше, ніж аналіз тих самих властивостей самостійно, без використання Visu.

Відобразимо гістограму середнього часу аналізу освітньої програми (рис. 6.1).

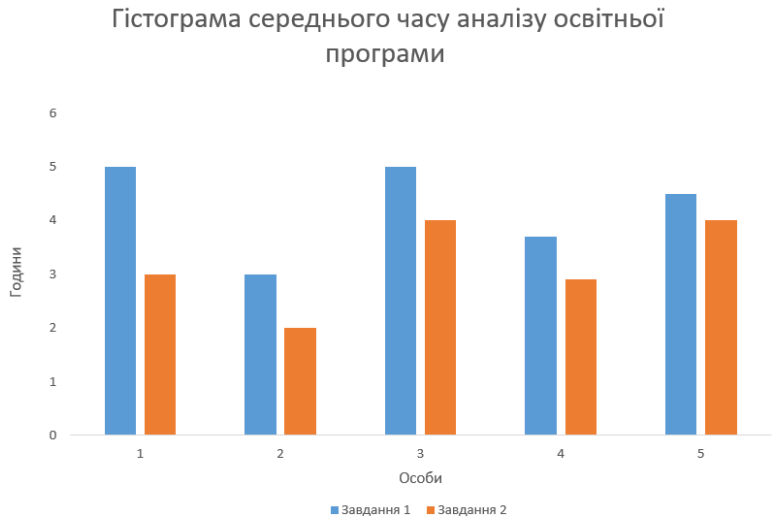

Рисунок 6.1 – Гістограма середнього часу аналізу освітньої програми

На цій гістограмі зображено одночасно два стовпця для кожної особи, що є зручним для їх порівняння: стовпці синього кольору показують середній час виконання першого завдання (аналізу властивостей освітньої програми без застосування програмних засобів), червоним кольором зображені стовпці, що відображають середній час виконання другого завдання (аналізу властивостей освітньої програми за допомогою програмного засобу Visu). По осі абсцис розташовані особи, а по ординат – години, витрачені на виконання завдання.

На рисунку 6.1 видно, що стовпці на гістограмі червоного кольору нижчі за відповідні стовпці синього кольору, що означає, що в середньому для виконання другого завдання витрачається менше часу у п’яти осіб, ніж для виконання першого завдання.

У даному розділі було проведено тестування програмного засобу Visu, де були складені два тестових випадки для двох варіантів використання, описаний процес тестування (тестування проводилося «чорним ящиком» та «білим ящиком»), приведений загальний відсоток покриття тестами програмної системи. Також була перевірена ефективність програмного засобу за допомогою експерименту, в якому брали участь п’ять осіб, що аналізували одні і ті ж властивості освітньої програми за чотирма критеріями два рази – без програмних засобів і за допомогою програмного засобу Visu. Критеріями були: порахувати які результати навчання були в кожній дисципліні кожного семестру в одному навчальному році, скільки результатів навчання було в кожній дисципліні в кожному семестрі, яких результатів навчання з усіх семестрів зустрічалося найчастіше, які результати навчання взагалі були в кожному семестрі. Також у цьому розділі була наведена гістограма середнього часу аналізу освітньої програми для обох разів.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці програмного засобу Visu, мета якого зменшити кількість годин, витрачених на аналіз властивостей освітньої програми, а саме відношенню дисциплін до результатів навчання на протязі семестрів за допомогою візуалізації даних у вигляді чотирьох діаграм.

Проведений аналіз предметної області освітньої програми показав, що після створення або затвердження освітньої програми можуть виникнути щонайменше три проблеми, які мають вплив на її якість. Для вирішення цих проблем найкращим рішенням є візуалізація властивостей освітньої програми у вигляді різноманітних діаграм, графіків тощо. На даний час існують багато програмних рішень для візуалізації даних, але в роботі було описано детально три найбільш часто використовуваних, з якими і порівнювався програмний засіб Visu. Перевагами Visu над наведеними аналогами є те, що він безкоштовний, підтримує файлове завантаження даних, а також розроблений програмний засіб проводить розрахунки перед відображенням кінцевого результату у вигляді діаграми користувачу на екран.

Створений програмний засіб Visu пройшов через весь життєвий цикл розробки програмного забезпечення, а саме: визначення проблем предметної області; формування функціональних і нефункціональних вимог до засобу; проектування програмного засобу, під час якого була обрана архітектура та технології розробки, створені допоміжні UML-діаграми у вигляді програмних класів та діаграм послідовності для декількох варіантів використання, схеми алгоритмів для реалізації функцій Visu; написання програмного коду, створення таблиць для бази даних програмного засобу та запитів для роботи з табличними даними; покриття програмного засобу на 90% тестами.

Як результат, вдалося створити програмне рішення, яке надає можливість з його використанням провести аналіз властивостей освітньої програми на 1.07 годину швидше, ніж аналіз тих самих властивостей самостійно, без використання Visu.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Стаття про освітню програму [Електронний ресурс] – Режим доступу: https://mon.gov.ua/ua/osvita/

2. Phil Simon. The Visual Organization: Data Visualization, Big Data, and the Quest for Better Decisions, 1st edition: Wiley, 2014. - 657 p.

3. Liubchenko, V., Komleva, N., Zinovatna, S., Pysarenko, K. Framework for systematization of data science methods // Applied Aspects of Information Technology. – 2021. – Vol. 4 № 1. – P. 80–90. DOI: 10.15276/aait.01.2021.7.

4. Maciaszek, L. Requirements Analysis and Systems Design / L. Maciaszek. Pearson Education Canada, 2007. – 656 p.

5. Jeffrey O. Grady. System Requirements Analysis, 2nd edition: Elsevier, 2014. – 553 p.

6. П. Клементс, Л. Басс, Р.Кацман. Архитектура программного обеспечения на практике, 2-ое издание: Питер, 2006. – 574 с.

7. Mark Richards, Neal Ford. Fundamentals of Software Architecture, 1st edition: O'Reilly Media, 2020. – 809 p.

8. Robert C. Martin. Clean Architecture: A Graftsman`s Guide to Sofware Structure and Design, 1st edition: Prentice Hall, 2017. – 1432 p.

9. Grady Booch. Unified Modeling Language User Guide, 2nd edition: Addison-Wesley Professional, 2017. – 496 p.

10. Кормен Т. Алгоритмы: построение и анализ, 3-е издание: Вильямс, 2016. – 1328 с.

11. Лайза К., Джанет Г. Гибкое тестирование – Практическое руководство для тестировщиков ПО и гибких команд, 1-ое издание: Вильямс, 2010. – 464 с.

12. Boris Beizer. Software Testing Techniques, 2nd Edition: Dreamtech, 2003. – 550 p.