Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: Розробка мультиплатформеної системи
для нотування та зберігання даних на основі графів

Виконав: студент ___4___ курсу
групи __1ПІ-20б____
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Грицина В.В.
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Ткаченко О.М.
(прізвище та ініціали)

Рецензент:. к.т.н., доц. каф ОТ Богомолов С. В.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ _О. Н. Романюк _
«_10_» __червня_____ 2024 р.

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 12 » березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Грицині Владиславу Вікторовичу

1. Тема роботи – розробка мультиплатформеної системи для нотування та зберігання даних на основі графів.

Керівник роботи: Ткаченко Олександр Миколайович, д-р техн. наук, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Строк подання студентом роботи 3 червня 2024

3. Вихідні дані до роботи: базові методи заповнення тексту, методи створення файлових нотаток на основі теорії графів, методи пошуку по графу Depth-first search та Breadth-first search, максимальна кількість тексту в файлах не обмежена, дані та координати про розташування вершин та граней у результуючому графів, кінцеве зображення мапи нотаток та їх зв'язків.

4. Зміст текстової частини: теорія графів; алгоритм пошук у глибину; алгоритм пошуку в ширину; розрахунок освітлення; оптичні властивості матеріалів; аналіз систем нотування з використанням графів; аналіз методів нотування даних на основі графів; алгоритми роботи з графами; алгоритм відображення графу; розробка інтерфейсу програми; розробка модуля відображення; розробка модуля зміни файлів; результат тестування системи; розробка структури програми; розробка інструкцій користувача;

5. Перелік ілюстративної частини: галузі застосування систем нотатків; основні етапи алгоритму для роботи з графами; алгоритми для пошуку в графі; здатностей поверхонь; інтерфейс роботи з нотатками; інтерфейс відображення графу.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ткаченко О.М., к.т.н., доц каф ПЗ	13.03.2024	03.06.2024

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз розвитку систем нотування	14.03.24 - 31.03.24	Вик.
2	Розробка методів нотування на основі теорії графів	01.04.24 - 18.04.24	Вик.
3	Розробка алгоритмів роботи з нотатками та інтерфейсу системи	19.04.24 - 06.05.24	Вик.
4	Тестування системи	07.05.24 - 24.05.24	Вик.
5	Оформлення матеріалів до захисту БДР	25.05.24 - 30.05.24	Вик.

Студент _____ **В.В. Гри**
(підпис) (ініціали та пр

Керівник бакалаврської дипломної роботи _____ **О. М. Ткач**
(підпис) (ініціали та пр

Анотація

УДК 004.912.032.26

Грицина В.В. Розробка мультиплатформеної системи для нотування та зберігання даних на основі графів: бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 65 с.

На укр. мові. Бібліогр. : 22 назв ; рис. : 37; табл. 8.

У бакалаврській дипломній роботі роботі розроблено програмні засоби алгоритми та структури даних для теми “Розробка мультиплатформеної системи для нотування та зберігання даних на основі графів”, обґрунтовано актуальність теми та доцільність розробки даної системи, крім того було проведено тестування програми для виявлення недоліків та їх усунення.

Було використано такі технології для реалізації системи як: мова програмування C# , фреймворк .NET з його платформою для розробки мультиплатформених програм MAUI, та Azure File Storage для зберігання нотаток.

Реалізовано систему та її частини, які надає можливість зберігати нотатки та дані у вигляді графів, та відображувати їх у вигляді карти графу.

Annotation

Hrytsina V.V. Development of a multi-platform system for notation and data storage based on graphs: Bachelor's thesis in specialty 121 Software Engineering, educational program - software engineering. Vinnytsia: VNTU, 2024. 65 p.

In Ukrainian. Bibliography: 22 titles; figures: 37; tables: 8.

Software tools, algorithms, and data structures were developed in the bachelor thesis for the topic " Development of a multi-platform system for notation and data storage based on graphs ". The relevance of the topic and the feasibility of developing this system were substantiated, and testing of the program was conducted to identify shortcomings and address them.

Technologies such as the C# programming language, the .NET framework with its MAUI platform for developing cross-platform applications were utilized for implementing the system.

The system and its parts have been implemented, which provides an opportunity to save notes and data in the form of graphs, and to display them in the form of a graph map.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. У сучасному світі обробка та аналіз даних стають все важливішими, особливо з огляду на зростаючі обсяги інформації. Використання графових баз даних набуває популярності через їхню здатність ефективно моделювати зв'язки між даними. Існують системи для зберігання та обробки даних, але багато з них мають обмеження, зокрема щодо складних структур даних.

Графове зберігання даних дозволить подолати ці обмеження, а мультиплатформеність таких систем допоможе працювати з даними на будь-якому пристрої [1]. Система на основі графів корисна у будь-яких областях зберігання та відображення даних в графічному вигляді, зокрема для нотаток і створення нових ідей на основі зв’язків між існуючими нотатками. Розробка такої системи допоможе систематизувати дані, покращить розуміння зв'язків між ними і створить можливість для отримання нових даних на їх основі.

Розробка даної системи є складним завданням, через необхідність розробки алгоритмів для ефективної роботи з даними, а також забезпечення високої швидкості роботи системи [2]. Важливим аспектом є також забезпечення безпеки даних, оскільки обробка великих обсягів інформації потребує надійного захисту від несанкціонованого доступу. Крім того, необхідно забезпечити зручний інтерфейс користувача для зручного використання системи.

Уже існують схожі системи, але в них є певні недоліки де-які з них не мають звичайної безкоштовної версії, деякі не надають можливостей мультиплатформеності, деякі не надають можливість створювати зв’язки між нотатками, таким чином, розробка даної системи є актуальною задачею.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою бакалаврської дипломної роботи є розширення функціональних можливостей ведення нотаток та перегляду їх зв’язків за рахунок застосування теорії графів, що дозволить користувачам систематизувати дані і створить передумови для отримання нових знань на їх основі.

Для досягнення поставленої мети необхідно розв’язати такі задачі:

- провести аналіз існуючих методів і засобів ведення нотаток на основі інших популярних алгоритмів;
- запропонувати та розробити загальний алгоритм системи;
- запропонувати та розробити покращений метод відображення графу на основі існуючих аналогів;
- реалізувати систему з використанням засобів графічного інтерфейсу для відображення даних на основі розроблених методів;
- здійснити тестування системи згідно поставлених задач.

Об’єкт дослідження. Об'єктом дослідження є процес зберігання та візуалізації даних.

Предмет дослідження. Предметом дослідження є методи, алгоритми та програмні засоби нотування, зберігання та візуалізації даних на основі теорії графів.

Методи дослідження. В процесі виконання роботи використовувалися теорія графів для створення алгоритмів для роботи з нотатками, методи обробки баз даних для створення сховища та алгоритмів роботи з сховищем для збереження нотаток, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів

1. Вдосконалено метод обходу графу в глибину (DFS), який відрізняється від існуючого можливістю обходу ізольованих вершин (гілок) за допомогою лінійного пошуку, що дозволило розширити сферу застосування DFS на графи, що містять кілька деревовидних структур або ізольовані вершини.

2. Вдосконалено метод візуалізації графу , в якому на відміну від існуючих, передбачено візуалізацію на основі послідовного перебору всіх вершин з присвоєнням випадкових координат кожній вершині та подальшого створення ребер з використанням зв’язків між вершинами, що дозволило отримати більш точну та зрозумілу візуалізацію структури графу та полегшує загальне розуміння взаємозв'язків між вершинами.

Практична значення одержаних результатів. Розроблено алгоритми та програми, за допомогою яких користувачі зможуть створювати нотатки та досліджувати їх. Алгоритми та окремі програмні модулі можуть бути використані розробниками для створення інших програмних систем для роботи з графами.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованих працях, опублікованих у співавторстві, автору належать такі результати: аналіз баз даних, аналіз методів можливої реалізації системи.

Апробація матеріалів бакалаврської дипломної роботи. Усі наукові результати, викладені у дипломній роботі, доповідалися на LIII Всеукраїнській науково-технічній конференції підрозділів Вінницького національного технічного університету (2024).

Публікації. Основні результати досліджень було опубліковано у матеріалах LIII Всеукраїнської науково-технічної конференції підрозділів Вінницького національного технічного університету (2024) [3].

1 АНАЛІЗ РОЗВИТКУ СИСТЕМ НОТУВАННЯ ТА ЗАСОБІВ ДЛЯ РОБОТИ З НИМИ

1.1 Аналіз стану систем нотування з використанням засобів для роботи з нотатками та файлами

З розвитком технологій, системи для нотування та зберігання даних на базі графів набувають все більшої популярності у різних галузях. Використання графових структур надає можливість ефективної організації та швидкого доступу до інформації, забезпечуючи нові можливості для оптимізації бізнес-процесів та аналізу даних.

Розробка програмних модулів для реалізації системи з використанням графів є ключовим завданням у цьому контексті, оскільки вона сприяє створенню інноваційних рішень, що поліпшують продуктивність та надають конкурентні переваги бізнесу. Однією з ключових переваг використання графових систем є їхня здатність моделювати складні зв'язки між об'єктами та даними, що дозволяє отримувати глибше розуміння інформації та приймати обґрунтовані рішення.

Компанії, щоб вирішити це завдання, звертаються до спеціалізованих розробників програмного забезпечення, які мають досвід у роботі з графовими технологіями. Ці фахівці можуть розробити програмні модулі, які оптимізовані під конкретні потреби бізнесу та забезпечують ефективне використання графових можливостей. Аутсорсинг такого завдання дозволяє компаніям зосередитися на своїх стратегічних цілях, заощаджуючи при цьому час та ресурси.

Інтеграція різноманітних функцій та можливостей в системи для нотування та зберігання даних на базі графів також має свої переваги. Наприклад, програмні модулі можуть включати в себе алгоритми аналізу графів [4], візуалізацію даних, а також інструменти для роботи з великими обсягами

інформації. Це дозволяє компаніям створювати потужні та гнучкі системи, які відповідають вимогам сучасного бізнесу та допомагають у прийнятті обґрунтованих рішень.

Отже, розробка програмних модулів для систем для нотування та зберігання даних на базі графів визнається критично важливою у сучасному цифровому ландшафті. Залучення спеціалізованих фахівців дозволяє компаніям створювати потужні та інноваційні рішення, які допомагають в оптимізації бізнес-процесів та забезпечують конкурентні переваги. З урахуванням швидкого розвитку технологій графових систем важливо відстежувати останні тенденції та застосовувати їх для підвищення ефективності та інноваційності.

1.2 Порівняльний аналіз аналогів

Використання різних технологій для створення програм та модулів для нотаток має велику кількість альтернатив, які мають велику кількість незвичайних унікальних функцій, за допомогою порівняння даних аналогів можна отримати, уявлення про сучасні можливості даних програм та створити нову систему яка буде мати покращенні, та вдалі функції з аналогів.

До найбільш відомих рішень відносять:

- Roam Research;
- Workflowy;
- Notion;
- Evernote.

Одним із прикладів програм для нотування є Roam Research (рис. 1.1) це інноваційний інструмент для структурування і редагування знань [5], який поєднує в собі особливості заміток, бази даних і процесу управління завданнями. За допомогою Roam користувач може створювати великі мережі зв'язаних нотаток, організовувати їх за допомогою хештегів і лінків, що дає можливість легко знаходити і аналізувати інформацію.

Відмінною рисою є можливість створення "блоків" тексту, які можна легко переміщувати, редагувати та повторно використовувати в інших частинах нотаток. Цей інструмент дозволяє користувачам організовувати свої думки за допомогою гнучких структур та швидко переходити від ідеї до її розробки та втілення, що робить його корисним для активних користувачів, які займаються навчанням, дослідженням або професійною діяльністю.

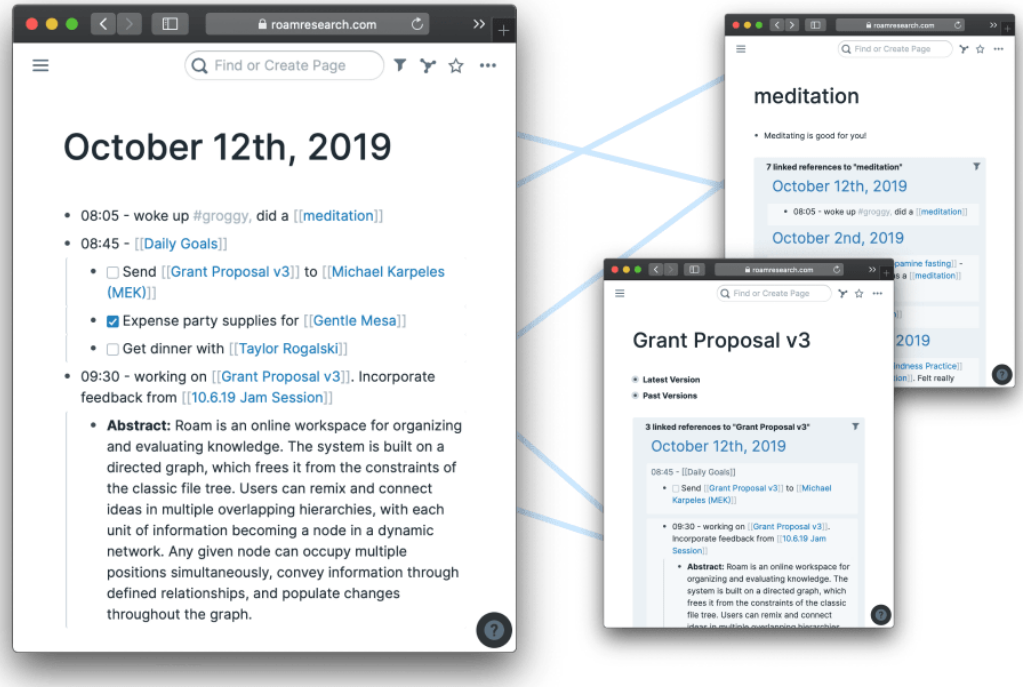


Рисунок 1.1 – Приклад роботи додатку Roam Research

Workflowy - це інструмент для створення списків та структурування ідей [6], який зосереджений на простоті та ефективності (рис. 1.2).

Його інтерфейс дозволяє користувачам створювати безліч списків, розгалужувати їх на рівні ієрархії, а також швидко переміщатися між різними рівнями деталізації. Workflowy забезпечує зручні інструменти для підсвічування, маркування та організації елементів списку, що дозволяє користувачам краще визначати пріоритети та робити замітки зрозумілішими.

Цей інструмент особливо корисний для осіб, які шукають простий спосіб управління завданнями, планування проектів або структурування інформації без зайвих складнощів.

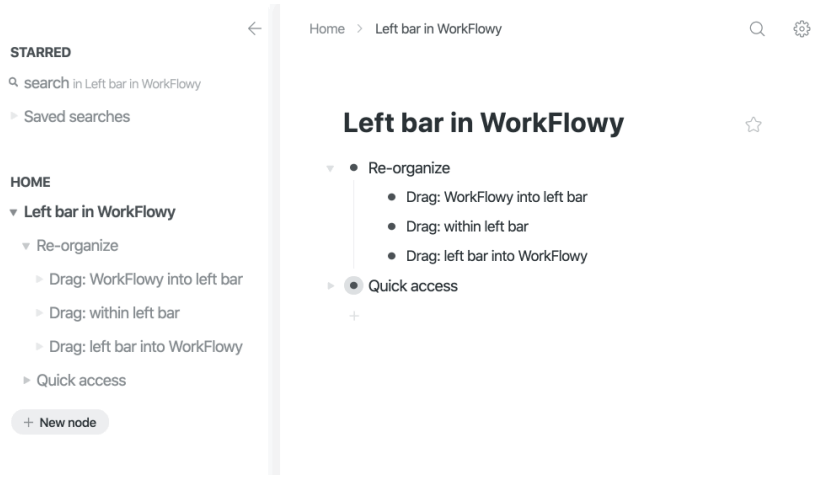


Рисунок 1.2 – Приклад роботи додатку Workflowy

Існує ще один такий застосунок як Notion (рис. 1.3), він не використовує графи [7], але є дуже відомим застосунком для зберігання нотаток, також Notion має величезний потенціал для організації проектів, спільної роботи команд та створення структурованих баз даних.

Він поєднує у собі можливості нотаток, календаря, задач і бази даних, що робить його універсальним інструментом для керування різноманітними аспектами життя та роботи.

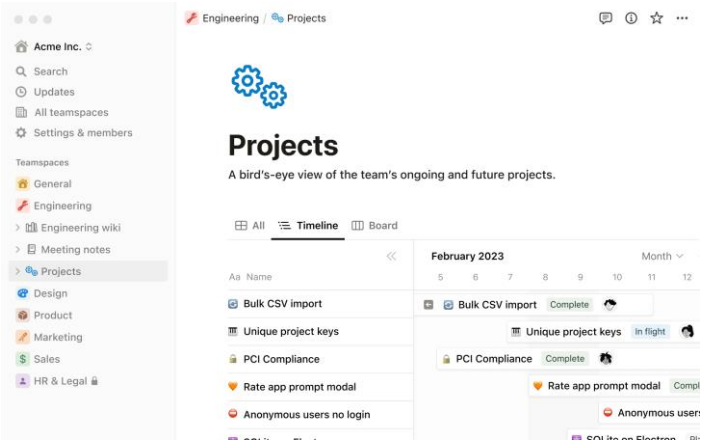


Рисунок 1.3 – Приклад роботи додатку Notion

Застосунок Evernote (рис. 1.3) – ще один популярний інструмент для зберігання нотаток, який вже довгий час займає своє місце серед найбільш використовуваних програм такого типу [8]. Evernote відомий своєю простотою використання та багатофункціональністю. Він дозволяє створювати різноманітні типи нотаток, включаючи тексти, голосові записи, фотографії та веб-сторінки.

Крім того, Evernote має можливості організації, такі як тегування, пошук та створення різних нотаткових блокнотів для різних потреб користувача. Це дозволяє зручно управляти великим обсягом інформації та швидко знаходити необхідні дані. Evernote також має функцію синхронізації між пристроями, що дозволяє користувачам легко працювати на різних пристроях та платформах.

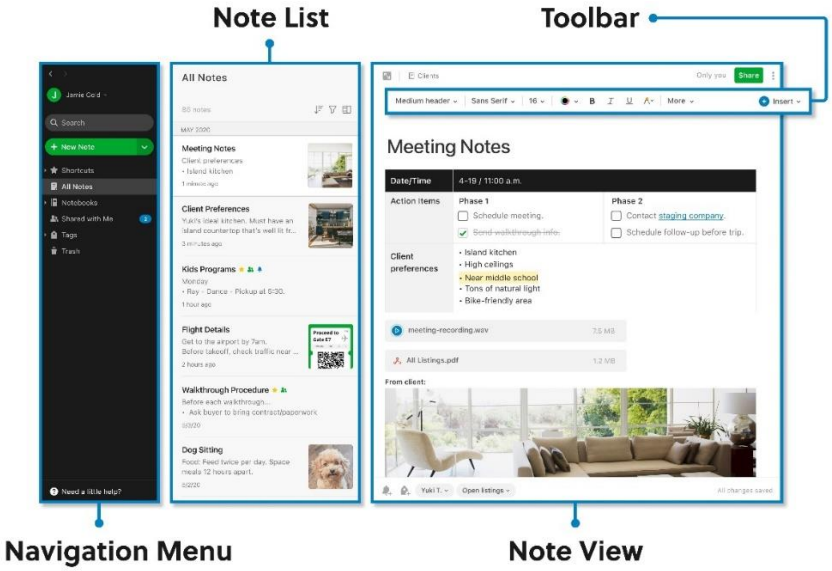


Рисунок 1.4 – Інтерфейс Evernote з прикладом роботи

Розробка системи для зберігання нотаток потребує використання різних інструментів та деякого досвіду який допоможе створити систему. Розробка даної система передбачає створення багатьох складових системи, не тільки візуального інтерфейсу для користувача, а також і бази даних або іншої

структури для зберігання нотаток у вигляді графів, так і відповідного модуля для роботи з даної базою даних та роботи з відправкою та отриманням даних.

Згідно наведених вище аналогів їх аналізу було створено таблицю для порівняння аналогів.

Результати порівняння аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльні характеристика онлайн платформ

Критерії	Roam Research	Workflowy	Notion	Evernote	Graph Notes
Можливість вбудованого пошуку	+	+	+	+	+
Можливість створення зв’язків між нотатками	+	-	-	+	+
Мультиплатформеність	-	+	+	+	+
Можливість роботи в офлайн режимі	-	-	+	-	+
Мобільний додаток	+	+	+	+	+
Загальна оцінка	60%	60%	80%	80%	100%

Відповідно до таблиці порівняльних характеристик розробка власної системи, яка зможе покрити недоліки існуючих рішень та забезпечити новий досвід та можливості застосування графів та покращення існуючих способів зберігання нотаток.

Отже, розробка програмних модулів для впровадження системи для зберігання нотаток відкриває широкі можливості для користувачів, а саме для роботи на різних платформах різним групам користувачів, користувачі зможуть зберігати та досліджувати власні нотатки використовуючи алгоритми побудовані на графах.

1.3 Аналіз методів розв’язання задачі

Розробка програмних модулів для створення мультиплатформених систем з використанням графів вимагає уважного вивчення оптимальних підходів до цього завдання. Існують різноманітні методи, кожен з яких має свої переваги та недоліки. У цьому розділі буде проаналізовано найбільш ефективні методи для розробки таких програмних модулів.

Одним з рішень є створення багатьох мало пов’язаних один між одним модулів які будуть разом створювати працюючу екосистему яка загалом і стане готовим застосунком для кінцевого користувача, але даний підхід не є оптимальним через важкість створення великої кількості сервісів, це потребує великої команди та правильного менеджменту створених модулів.

Іншим варіантом є створення однієї монолітної системи яка буде мати в собі всі потрібні елементи, буде включати в себе всі потрібні складові функції бібліотеки та інші інструменти, це й підхід не підходить до даного проєкту через потрібність зберігання даних не тільки на стороні користувача для передачі та захисту даних користувача.

Існує також третій варіант, який може бути більш оптимальним для розробки програмних модулів для мульиплатформкних систем. Це використання архітектури мікросервісів.

Мікросервісна архітектура полягає в тому, щоб розділити функціонал програмного забезпечення на невеликі незалежні сервіси, які працюють разом

через API. Кожен сервіс відповідає за обмежену функціональність і може бути розгорнутий та масштабований незалежно від інших.

В контексті розробки систем, це означає, що різні аспекти функціональності, такі як обробка зображень, взаємодія з геолокацією, обробка даних користувача та інші, можуть бути реалізовані як окремі мікросервіси. Це полегшує розробку, тестування та підтримку системи, оскільки кожен сервіс може бути розроблений незалежно, а також масштабований окремо в разі потреби.

Звичайно, використання мікросервісної архітектури також має свої виклики, такі як складність управління багатьма сервісами та підтримка консистентності даних між ними. Однак, з правильним плануванням і реалізацією, цей підхід може забезпечити більшу гнучкість та масштабованість для розробки систем.

З вище наведених варіантів випливає останній найбільш логічний і доцільний підхід, а саме використання гібриду двох вище наведених підходів. Це буде реалізовано за допомогою створення окремого модулю для роботи з базою даних та набору застосунків які будуть виконувати роль інтерфейсів користувача, кожен на відповідній платформі.

1.4 Постановка задач дипломної роботи

Після ретельного аналізу переваг і недоліків наявних систем записування нотаток отримано висновок, що ці методи хоча і мають свої переваги, але також істотні недоліки, які вимагають уваги.

Одним з найбільш важливих недоліків є їх неоднозначність. Часто у системах записування нотаток можуть виникати ситуації, коли одна і та ж сама символіка або конструкція може мати різні тлумачення в залежності від контексту або індивідуального розуміння користувача. Це ускладнює

використання системи та може призводити до непорозумінь або помилкового інтерпретування нотаток.

Ще одним недоліком є недостатня гнучкість для вираження різноманітності зберігання ідей. Багато існуючих систем записування нотаток обмежені у своїх можливостях для вираження складних або нетрадиційних концепцій. Це може бути обмеженням для творчості та інновацій, оскільки користувачі можуть відчувати себе обмеженими у виразній силі своїх нотаток.

Крім того, існуючі системи мають обмежену можливість передачі емоцій та інтенцій через записи. Хоча слова та символи можуть передавати певну інформацію, вони часто не можуть відобразити повністю емоційне забарвлення або наміри автора записів, що може призводити до втрати частини виразності та інтерпретації інформації.

Отже, вирішення цих недоліків є ключовим завданням у розробці нових методів записування нотаток, які будуть більш ефективними та точними у вираженні ідей та емоцій через записи.

З чого випливають наступні завдання, які необхідно виконати для розробки власної системи:

- провести аналіз існуючих методів і засобів ведення нотаток на основі інших популярних методів;
- запропонувати та розробити загальний алгоритм системи;
- запропонувати та розробити покращений метод відображення графу на основі існуючих аналогів;
- реалізувати систему з використанням засобів графічного інтерфейсу для відображення даних на основі розроблених методів;
- здійснити тестування системи згідно поставлених задач.

1.5 Висновки

У першому розділі дослідження було проаналізовано поточнний стан систем для ведення нотаток, які широко використовуються в даний час. Для цього було ретельно досліджено кілька провідних платформ, таких як Roam Research, Workflowy, Notion та Evernote, щоб з'ясувати їхню популярність серед користувачів та визначити їхні переваги та обмеження.

Під час аналізу було докладно розглянуто, які саме аспекти зручності використання та доступності на різних пристроях роблять ці платформи особливо привабливими для користувачів.

Крім того досліджено, які саме аспекти зручності використання найбільше цікавлять користувачів, чи це можливість швидкого створення та редагування нотаток, зручний пошук і організація матеріалів, або можливість спільної роботи та обміну нотатками з іншими користувачами.

Однак, при докладному розгляді було помічено, що деякі з них не відповідають усім потребам користувачів. Наприклад, деякі можуть бути обмежені у функціональності, інші не мають підтримки для всіх платформ або можуть бути складними у використанні.

Враховуючи ці обмеження отримано висновок, що розробка власної системи нотаток є доцільною задачею. Це дозволить створити програмне рішення, яке відповідає конкретним потребам наших користувачів, забезпечуючи при цьому необхідний функціонал та мультиплатформенність.

Визначено основні завдання для успішної реалізації цього проекту, включають в себе розробку повного спектру функцій, які відповідають потребам користувачів, створення інтуїтивно зрозумілого і зручного інтерфейсу, забезпечення безпеки даних та їхньої синхронізації між пристроями, а також розширюваність системи для майбутніх покращень та інтеграцій з іншими сервісами.

2 ТЕОРЕТИЧНЕ ЗАСТОСУВАННЯ ТА ВДОСКОНАЛЕННЯ МЕТОДІВ ПРОГРАМНОГО ЗАСТОСУНКУ

2.1 Застосування методів теорії графів для нотування даних

На основі проведеного у р. 1 аналізу для формального опису нотування та візуалізації даних обрано теорію графів. Граф представляється як впорядкована пара G:=(V,E), де V - множина вершин, а E - множина (у випадку неорієнтованого графа - невпорядкованих) пар вершин з V, які називаються ребрами. Часто V і E вважають скінченними множинами, оскільки багато результатів, що стосуються скінченних графів, можуть бути неправильними або непридатними для нескінченних графів.

Степінь вершини у графі – це кількість ребер, що йдуть з цієї вершини. Для неорієнтованого графа степінь вершини відображається як кількість ребер, що прилягають до цієї вершини, тобто кількість сусідніх вершин.

З вищезазначеного опису графу, нотатки та зв'язки між ними можна утворити граф, що випливам чином відповідний алгоритм буде перетворювати ці посилання на зв’язки є з визначення графу, його вершин і граней.

В якості зв’язків будуть використовуватись посилання в тексті на інші нотатки між даним графом та іншими графами.

Існує велика кількість різних алгоритмів для збереження даних на основі графів, починаючи від класичних, таких як алгоритми обходу графів в глибину або в ширину, і до сучасних глибоких нейронних мереж, які можуть вивчати складні залежності в графових структурах. Такі алгоритми можуть бути використані в різних галузях, включаючи соціальні мережі, біоінформатику, транспортні системи та багато інших.

Для детального аналізу було обрано всього декілька з них а саме:

- матричне нотування;
- спискове нотування;

- списокове нотування зважених графів;
- матричне нотування зважених графів.

Нижче наведено їхній більш детальний опис, з загальним описом роботи алгоритму, його недоліками та позитивними сторонами.

Матричне нотування графів - це метод, при якому граф подається у вигляді квадратної матриці. У такій матриці рядки та стовпці відповідають вершинам графа, а значення в кожній комірці вказує на наявність (або відсутність) зв'язку між відповідними вершинами. Якщо вершини iі та jj з'єднані, то відповідний елемент aijаij матриці приймає значення 1, інакше - 0. Цей метод особливо ефективний для графів з малою кількістю ребер, оскільки в матриці буде багато нульових елементів, що дозволяє зберігати граф у компактному вигляді та ефективно використовувати пам'ять (рис. 2.1).

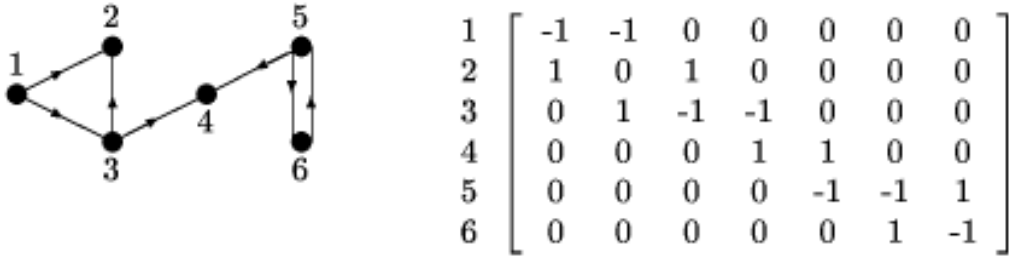


Рисунок 2.1 – Граф, з матричною таблицею

Проте, для графів з великою кількістю вершин або з багатою кількістю ребер матричне нотування може бути неефективним з точки зору використання пам'яті. У разі розріджених графів, коли більшість елементів матриці будуть нульовими, цей метод може призвести до значного розроблення пам'яті. Крім того, додавання або видалення вершин у графі може вимагати зміни розмірності матриці, що може бути складним з точки зору реалізації та вимагати багато часу на обчислення.

Спискове нотування графів - це метод, при якому кожна вершина графа зберігається у списку разом із списком вершин, до яких вона має прямі зв'язки. Кожен елемент у списку вершин вказує на зв'язок між поточною вершиною та вершиною зі списку. Цей метод дозволяє ефективно використовувати пам'ять, особливо для розріджених графів, оскільки зберігає тільки необхідну інформацію про зв'язки між вершинами (рис. 2.2). Додавання або видалення вершин зазвичай здійснюється шляхом додавання чи видалення елементів у відповідних списках.

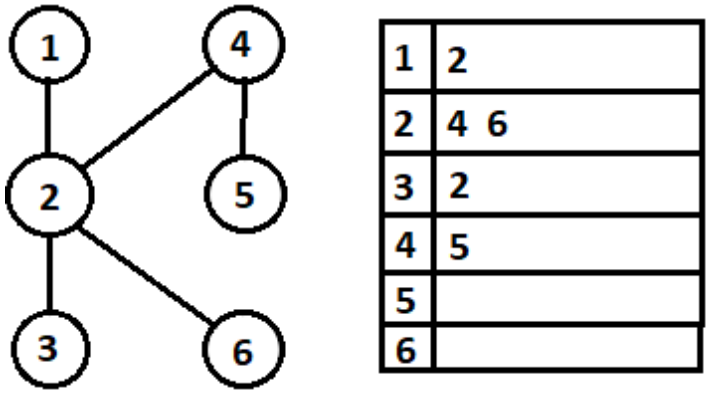


Рисунок 2.2 – Граф з списком зв’язків.

Проте, для графів з великою кількістю вершин, але з обмеженою кількістю ребер, спискове нотування може бути ефективнішим з точки зору використання пам'яті, оскільки зберігає тільки необхідну інформацію про зв'язки. Однак пошук зв'язків між вершинами може вимагати додаткового часу у порівнянні з матричним нотуванням, особливо для графів з великою кількістю ребер. Додавання чи видалення ребер також може бути більш складним у списковому нотуванні, оскільки потребує зміни структури списків.

Спискове нотування зважених графів розширює базовий підхід спискового нотування, додавши можливість зберігання ваги кожного ребра. Кожна вершина графа все ще має свій список суміжних вершин, але тепер кожен елемент цього

списку також містить інформацію про вагу зв'язку між поточною вершиною і вершиною зі списку. Цей метод особливо корисний для алгоритмів, які вимагають інформацію про вагу ребер, таких як алгоритми пошуку найкоротших шляхів (наприклад, алгоритм Дейкстри або алгоритм Беллмана-Форда).

Переваги спискового нотування зважених графів включають ефективне використання пам'яті для розріджених графів і збереження інформації про вагу ребер. Цей метод дозволяє легко отримувати доступ до ваги кожного ребра, що дуже корисно для виконання різних алгоритмів на графах.

Проте, як і у випадку зі звичайним списковим нотуванням, пошук зв'язків між вершинами може бути менш ефективним у порівнянні з матричним нотуванням, особливо для графів з великою кількістю ребер. Крім того, додавання або видалення ребер може вимагати модифікації структури списків, що може бути складним у реалізації і вимагати додаткового часу на операції зміни структури даних.

Після аналізу всіх трьох наведених алгоритмів було вирішено скористатися списковим нотуванням графів. Це обрано через те, що графи матимуть список з посиланнями на інші графи, що дозволить графам ефективно шукати зв'язки між собою та зберігати пам'ять. Крім того, цей метод забезпечує ефективність операцій пошуку та маніпулювання графами.

Також, завдяки можливості нотування ізольованих вершин графів, що не мають зв'язків з іншими вершинами, можливо зберігати всі вершини в загальному списку для забезпечення доступу до ізольованих вершин та операцій з ними, що б уникнути майбутніх проблеми з ізольованими вершинами та доступом до них.

2.2 Вдосконалений метод обходу графу та його застосування для нотування даних

Після реалізації алгоритмів для графового нотування, потрібно буде також мати відповідні методи для роботи з створеною графою структурою, основним методом буде один із методів пошуку на його основі і будуть реалізовані операції отримання відповідного графу, зміни та видалення.

В якості можливих методів для пошуку можна використати BFS та DFS.

DFS (Depth-First Search) – це метод обходу графа, який починається з однієї вершини і продовжує рухатися "вглиб" графа, доки не буде досягнуто кінцевої вершини або поки не будуть відвідані всі вершини [9]. Основна ідея полягає в тому, щоб спочатку відвідати всі сусідні вершини поточної, а потім перейти до наступної неперевіреної вершини.

Основні кроки методу DFS:

1. Вибір стартової вершини: Починаючи з вибраної стартової вершини, починаємо обхід.
2. Відвідування вершини: Відвідуємо поточну вершину і позначаємо її як відвідану.
3. Перехід до сусідів: Обираємо одну з невідвіданих сусідніх вершин і повторюємо процес для неї. Якщо всі сусідні вершини вже відвідані, або ж немає сусідів, які ще не були відвідані, виконання повертається до попередньої вершини.
4. Повторення: Процес повторюється, поки всі вершини не будуть відвідані.
5. Завершення: Обхід графа завершується, коли всі вершини будуть відвідані.

Метод DFS може бути реалізований як рекурсивно, так і ітеративно. У рекурсивному варіанті використовується стек викликів функцій для збереження поточного стану обходу, а в ітеративному – стек даних для збереження вершин, які ще потрібно відвідати.

DFS часто використовується для розв'язання таких задач, як пошук шляху в графі, визначення наявності циклів, виявлення зв'язних компонентів та інші задачі, пов'язані з обробкою графів.

BFS (Breadth-First Search) або обхід в ширину – це метод обходу графа, який починається з однієї вершини і пропагується "вшир" графу, відвідуючи всі сусідні вершини на поточному рівні перед переходом до наступного рівня [10].

Основні кроки метод BFS:

1. Вибір стартової вершини: Починаємо з вибраної стартової вершини.
2. Відвідування вершини: Відвідуємо початкову вершину і додаємо її до черги.
3. Перехід до сусідів поточної вершини: Починаємо відвідувати всі сусідні вершини поточної вершини. Кожну сусідню вершину, яка ще не була відвідана, додаємо до черги.
4. Повторення для наступного рівня: Після відвідування всіх сусідніх вершин поточного рівня, переходимо до наступного рівня, вилучаючи вершину з початку черги.
5. Продовження обходу: Процес повторюється, поки черга не стане порожньою.
6. Завершення: Обхід графа завершується, коли всі вершини будуть відвідані.

Основними структурами даних, які використовуються в методі BFS, є черга (Queue) для збереження вершин, які чекають на обробку, та масив або список, який відстежує, які вершини вже були відвідані.

Однією з ключових переваг мотоду BFS є те, що він завжди знаходить найкоротший шлях між двома вершинами у невзваженому графі, якщо такий шлях існує. Це робить його корисним для задач, пов'язаних з пошуком шляху, таких як пошук шляху в мережах, аналіз соціальних графів тощо.

Взагалом що DFS, що BFS є хорошими методами для реалізації пошуку по графі, але використання DFS буде кращим у даному випадку, через те що

користувач системи, частіше буде шукати відповідний граф пошук в глибину може допомогти знайти відповідну вершину, саме за допомогою більш глибокого пошуку, окрім цього пошук в глибину буде використовувати менше пам’яті порівняно з пошуком в ширину, при правильній реалізації.

Отже в результаті було обрано DFS, але стандартний метод не може задовільнити дані проблеми, через особливість нотаток зі зв’язками як графової структури.

Існує 2 основні проблеми використання DFS, як методу для пошуку по графу нотаток:

1. Можливість зациклення зв’язків: Нотатки можуть мати посилання на будь-які інші посилання, в такому випадку будуть створюватись циклічні посилання між 3 та більше нотатками.
2. Існування ізольованих нотаток: Будь-яка нотатка має можливість не містити в собі посилань на інші нотатки, та жодна інша нотатка може не мати посилань на дану нотатку, в такому випадку нотатка буде ізольованою від усіх інших нотаток.

Для вирішення даних проблем потрібно покращити метод DFS, спочатку для вирішення за циклювання зв’язків буде використано структуру HashSet.

HashSet – це структура даних, яка представляє собою набір унікальних елементів без порядку, що базується на хеш-таблиці. Вона забезпечує ефективну вставку, видалення та перевірку присутності елементів за короткий час. У HashSet кожен елемент унікальний, тобто він може бути доданий лише один раз, і він не має порядку, в якому елементи були додані.

Існування ізольованих нотаток можна вирішити використавши звичайний лінійний пошук по нотаткам які не входять до вже перевірених які знаходяться в HashSet.

Отже, в результаті отримано результуючий метод на основі DFS, який використовує HashSet та лінійний пошук, для створення оптимальної функції пошуку, яка зможе шукати по ізольованих та повторюваних нотатках.

Основні кроки результуючого методу пошуку:

1. Вибір стартової вершини: Починаючи з вибраної стартової вершини, починаємо обхід.
2. Відвідування вершини: Відвідуємо поточну вершину і перевіряємо чи була вона відвідана, якщо ні додаємо її до HashSet, інакше пропускаємо дану вершину, і переходимо до наступної.
3. Перехід до сусідів: Обираємо одну з невідвіданих сусідніх вершин і повторюємо процес для неї. Якщо всі сусідні вершини вже відвідані, або ж немає сусідів, які ще не були відвідані, виконання повертається до попередньої вершини.
4. Повторення: Процес повторюється, поки всі вершини не будуть відвідані.
5. Завершення: Обхід графа завершується, коли всі вершини будуть відвідані.
6. Лінійний пошук: Потім повторюємо перший крок вибираючи іншу початкову вершину, яка не є наявною в HashSet.

Також буде використано ще один метод для пошуку, він буде оснований на сортуванні графів за допомогою степені графу, за допомогою даного пошуку можна буде пришвидшити пошук популярних графів, через їх велику кількість зв’язків тому що чим більша кількість зв’язків тим більший шанс того що користувачу потрібно буде саме цей граф.

2.3 Вдосконалений методу візуалізації графу

Візуалізація графу, це важкий процес який потребує значних обчислювальних ресурсів і ретельного налаштування методів для розташування вершин і ребер у зручний для аналізу вигляд. Такий процес включає в себе вибір правильних методів і інструментів для відображення графу та врахування різноманітних характеристик даних.

Першою важливою частиною даного методу є саме присвоєння координат вершинам, для візуалізації графу кожна вершина повинна мати певні координати на площині, які потім будуть відображатись за допомогою відповідних інструментів.

Наступною частиною яку потрібно взяти до уваги є зв’язки між вершинами в даному випадку, відомо що посилання в даних нотатках є зв’язками і використовуючи координати вершин можна візуалізувати прямі між двома вершинами, таким чином можливо легко зображати грані між вершинами.

Для кожної вершини початкові координати не важливі, важливі саме зв’язки завдяки цьому координати можна згенерувати випадково для кожної вершини на початку відображення, після створення координат відбувається перебирання всіх вершин та їх зв’язків.

Потім на координатах кожної вершини відображається круг, який відображає вершину та ім’я даної вершини, після чого відображаються лінії між координатами вершин що відображає зв’язки між ними (рис. 2.3).

Результуючий метод відображення графу буде мати нижче наведену послідовність дій:

1. Вибір вершини: Вибираємо вершину з списку вершин.
2. Присвоєння координат: Присвоєння випадкових координат для кожної вершини, між мінімальними та максимальними, розмірами вікна відображення.
3. Перехід до наступної вершини: Переходимо до наступної вершини.
4. Візуалізація вершин: Візуалізуємо кожну вершину у вигляді кола за допомогою відповідних інструментів, також підчас візуалізації додаємо кожен зв’язок до HashSet.
5. Візуалізація ліній: Візуалізуємо всі лінії за допомогою відповідних інструментів графічної бібліотеки, які присутні в даному HashSet.

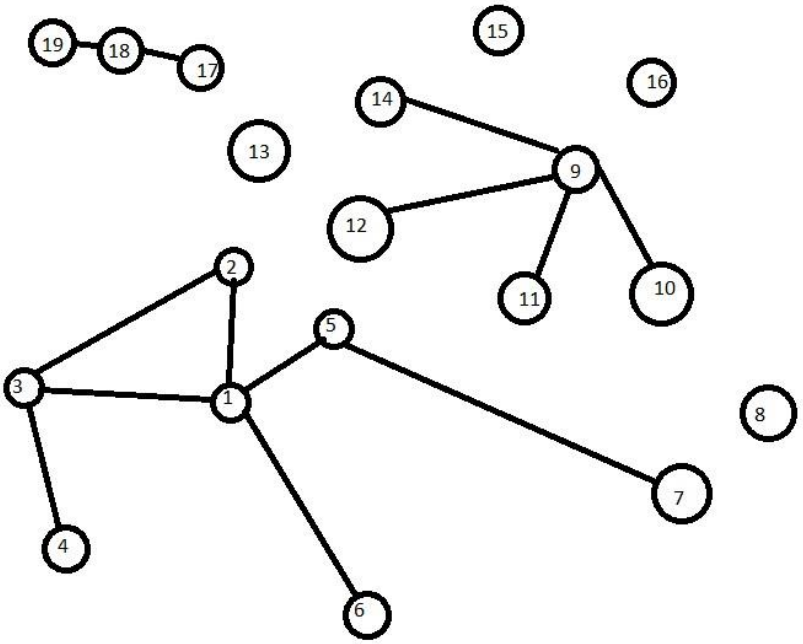


Рисунок 2.3 – Результуюче зображення графу

У результаті впровадження зазначеного методу відображення графу користувач отримає детальне 2D графове представлення своїх нотаток.

Це візуальне відображення відображає структуру та організацію нотаток у вигляді графу, що може значно полегшити їхнє сприйняття та розуміння користувачем.

Користувачі зможуть краще зрозуміти зв’язки між нотатками, їхню ієрархію та взаємозв’язки, що сприятиме більш ефективній навігації та управлінню нотатками.

Графове представлення дозволить швидше виявляти залежності між різними частинами проекту, зрозуміти логіку взаємодії та шляхи оптимізації. Це значно підвищить продуктивність та зручність роботи з нотатками для користувачів системи.

2.4 Висновки

У другому розділі роботи було запропоновано формальний опис процесу нотування даних із застосуванням апарату теорії графів. Вдосконалено метод обходу графу в глибину (DFS), який відрізняється від існуючого можливістю обходу ізольованих вершин (гілок) за допомогою лінійного пошуку, що дозволило розширити сферу застосування DFS на графи, що містять кілька деревовидних структур або ізольовані вершини.

Вдосконалено метод візуалізації графу, в якому на відміну від існуючих, передбачено візуалізацію на основі послідовного перебору всіх вершин з присвоєнням випадкових координат кожній вершині та подальшого створення ребер з використанням зв’язків між вершинами, що дозволило отримати більш точну та зрозумілу візуалізацію структури графу та полегшує загальне розуміння взаємозв'язків між вершинами.

Завдяки проведеному аналізу та обраних на його основі методів теорії графів, було вдосконалено існуючі методи для обходу, та візуалізації графу, завдяки чому користувач отримає нові можливості, для роботи з власними нотатками, досліджувати візуалізований граф та робити пошук по його нотаткам.

3 РОЗРОБКА СТРУКТУРИ, АЛГОРИТМІВ ТА МОДУЛІВ РЕАЛІЗАЦІЇ МУЛЬТИПЛАТФОРМЕНОЇ СИСТЕМИ З ВИКОРИСТАННЯМ ГРАФІВ

3.1 Аналіз вхідних даних та роботи системи

В якості вхідних даних користувач буде додавати свої нотатки, а саме це буде текст, списки таблиці, та інші текстові дані які він буде додавати через відповідний інтерфейс, також за допомогою спеціального функціоналу користувач буде додавати відповідні зв’язки між його нотатками.

Розробка даної програми поділена на модулі для полегшення та більш структурованої розробки. Першим модулем буде створення програмного інтерфейсу для додавання та зміни інформації та назв у нотатках. Цей модуль буде відповідати за створення нотаток та їх зміну, а також за знаходження посилань на інші нотатки.

Другий модуль складатиметься з функціоналу, призначеного для зберігання і відображення нотаток на сервері. Він буде взаємодіяти з попереднім модулем, який, наприклад, може виконувати завантаження нотаток на сервер. Основними функціями другого модуля будуть зберігання нотаток у відповідних каталогах на сервері та створення списку доступних нотаток для подальшого відображення.

Інтеграція з попереднім модулем дозволить автоматично додавати нові нотаток до списку та забезпечити коректне відображення актуальної інформації. Крім того, цей модуль може забезпечити можливість фільтрації та сортування нотаток за різними параметрами, які будуть корисні для зручного використання користувачами. В результаті користувачі матимуть можливість легко знаходити та переглядати необхідні нотатки зі списку, що буде відображений.

Крім основного функціоналу зберігання та відображення нотаток, другий модуль буде також відповідати за синхронізацію змін у нотатках та управління

одночасним доступом до них. Це означає, що якщо один користувач редагує нотатку, інші користувачі можуть бачити ці зміни у реальному часі.

Синхронізація змін дозволить користувачам спільно працювати над нотатками, не переймаючись про можливі конфлікти. Механізми контролю версій та блокування ресурсів можуть бути використані для уникнення ситуацій, коли два або більше користувачів намагаються змінити одну і ту саму нотатку одночасно.

Таким чином, другий модуль не лише забезпечує доступ до нотаток у вигляді списку, але і гарантує їхню актуальність та цілісність під час одночасної роботи з ними кількох користувачів.

Останнім і найважливішим модулем є графічне відображення. Розробка даного модуля надасть можливість відображати нотатки у двовимірному вигляді на площині за допомогою вершин та граней.

Розробка цих програмних модулів вимагає глибокого розуміння комп'ютерного зору, графічного програмування та програмування на .NET. Для реалізації цих модулів будуть використані різноманітні інструменти та бібліотеки програмування, зокрема Visual Studio та .NET. Програмні модулі будуть протестовані та вдосконалені для забезпечення безперебійної роботи з системою для користувачів.

Отже, дана розробка мультиплатформеної системи для нотування та зберігання даних на основі графів, є комплексним та важким завданням. Модулі мають бути ретельно спроектовані, реалізовані та протестовані для забезпечення ефективності та гарного досвіду роботи користувача.

3.2 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Розробляючи програмні модулі для мультиплатформеної системи, яка використовує графи, важливо ретельно обрати технології яку будуть використані

під час розробки даної системи, вони мають надавати можливість покрити багато платформ, при цьому бути довго підтримуваними легкими в освоєні та використанні, швидкими та захищеними, наведений вище перелік особливостей дуже важливий для створення сучасної надійної системи.

C# - це об'єктно-орієнтована мова програмування, яка розроблена компанією Microsoft (рис. 3.1). C# використовується для розробки різноманітних типів додатків, включаючи веб-додатки, мобільні додатки, десктопні програми, ігри та багато іншого [11]. Вона залишається однією з найпопулярніших мов програмування в індустрії завдяки своїй потужності, ефективності та простоті використання.

Рисунок 3.1 – Логотип мови програмування C#

.NET - це платформа розробки програмного забезпечення, яка надає розробникам масштабовану та ефективну інфраструктуру для створення різноманітних застосунків, включаючи мобільні додатки (рис. 3.2). Завдяки своїй потужній інструментальній базі та широкій підтримці спільноти, .NET забезпечує стабільну та надійну основу для розробки [12].

Рисунок 3.2 – Логотип мови програмування C# з фреймворком .NET

MAUI (Multi-platform App UI) – це нова мультиплатформена ініціатива в рамках .NET, яка дозволяє розробникам створювати мобільні додатки, які працюють на різних платформах, таких як iOS, Android і Windows, з використанням одного коду (рис. 3.3). MAUI спрощує розробку, оскільки розробники можуть використовувати звичні інструменти та мови програмування .NET для створення додатків, які працюють на різних пристроях [13].

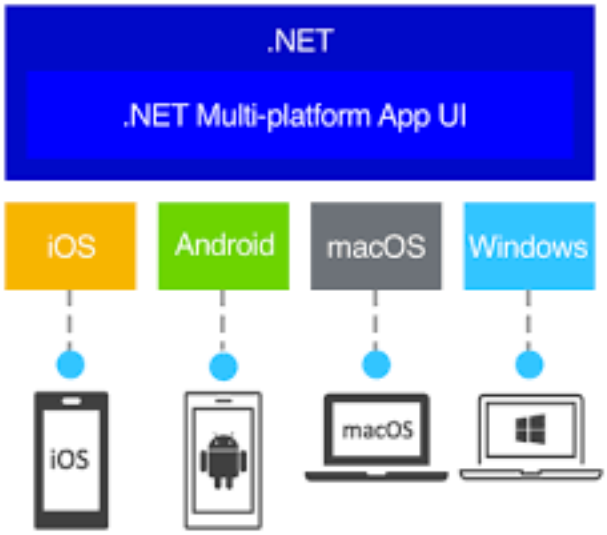


Рисунок 3.3 – Платформи підтримуванні .NET MAUI

При обиранні технологій для мультиплатформеної системи, яка використовує графічні інтерфейси, важливо враховувати не лише потужність та ефективність платформи, але й її здатність до швидкого розвитку та пристосування до змін у вимогах ринку. C# з використанням .NET і MAUI

представляють собою потужну комбінацію інструментів, яка задовільняє ці критерії, в якості середовища розробки взято Visual Studio [14].

Обрано Azure File Storage service як сховище для збереження даних з-за кількох переваг (рис. 3.4). Azure є однією з провідних платформ хмарних обчислень, яка пропонує надійні, масштабовані та безпечні послуги. File Storage service від Azure надає простий та ефективний спосіб зберігання та керування файлами в хмарному середовищі.

Він дозволяє легко масштабувати обсяги даних в залежності від потреб проекту і забезпечує високу доступність та надійність даних завдяки розподіленій архітектурі. Крім того, використання Azure File Storage спрощує інтеграцію з іншими послугами та рішеннями, що надаються Azure, такими як обчислювальні вузли, мережеві сервіси та інструменти аналізу даних.

Рисунок 3.4 – Логотип Azure File Storage

Обираючи C#, .NET та MAUI для мультиплатформенної системи, потрібні надійні та потужні інструменти для розробки програмного забезпечення, яке здатне працювати на різних пристроях, включаючи комп'ютери, смартфони та планшети. C#, яка є мовою програмування в рамках платформи .NET, надає зручний та ефективний інструментарій для створення якісного програмного забезпечення з високою продуктивністю та безпекою. Фреймворк .NET дозволяє

розробникам ефективно використовувати загальну кодову базу для різних платформ, що спрощує розробку та підтримку додатків.

MAUI (Multi-platform App UI) є інструментом, який розширює можливості розробки на основі .NET, дозволяючи створювати один додаток, який може працювати на різних платформах, таких як Android, iOS, macOS та Windows. Це дозволяє зберегти час та зусилля, які зазвичай потрібні на створення окремих додатків для кожної платформи.

Щодо Azure File Storage, використання цієї послуги дозволяє забезпечити синхронізацію даних між різними пристроями користувача, що робить процес роботи з додатком більш зручним та ефективним. Крім того, Azure File Storage забезпечує високу доступність та безпеку даних, що дозволяє зберігати і керувати інформацією без турбот про втрату чи порушення конфіденційності.

Що загалом надає потрібні інструменти для створення сучасної ефективної надійної та мільтиплатформеної системи, яка зможе реалізувати всі визначенні завдання.

3.3 Розробка інтерфейсу програмного додатку

Розробляючи інтерфейс даної системи вирішено з робити його прости і ефективним в використанні за для легкого використання на будь-яких платформах, та схожості інтерфейсу для легшої адаптації користувача.

В якості кольору було обрано інтерфейс за замовчуванням по причині простоті та лаконічності інтерфейсу, а саме темний колір з відтінками блакитного та фіолетового, генерація графу вирішено зробити темним та білим кольором для

Основним кольором інтерфейсу обрано темно-сірий, оскільки даний колір є нейтральним. Допоміжними кольорами обрано блакитний та білий, оскільки вони влучно контрастують один одному [15].

При відкритті додатку, користувач одразу бачить екран із інтерфейсом та сторінку з логотипом (рис. 3.5), з якої користувач почне використання додатку, це зроблено для більш ефективного та швидкого використання додатку.

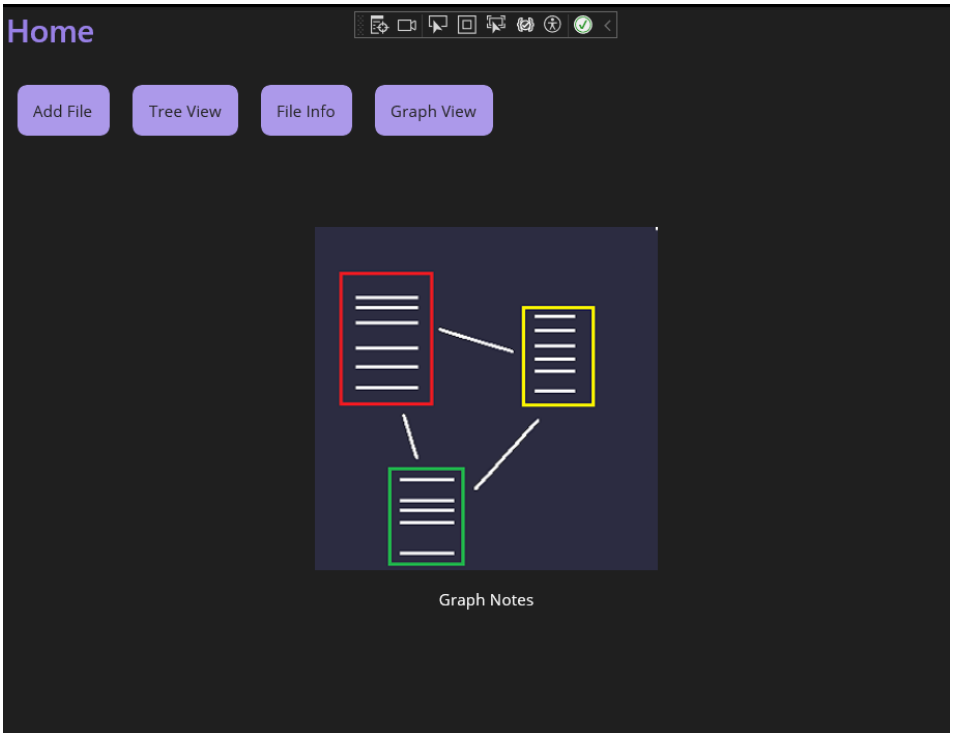


Рисунок 3.5 – Екран початку роботи програми

Додаток має зверху кнопки меню які можуть бути використані для навігації по додатку, тобто для переходу до відповідної частини системи (рис. 3.6).

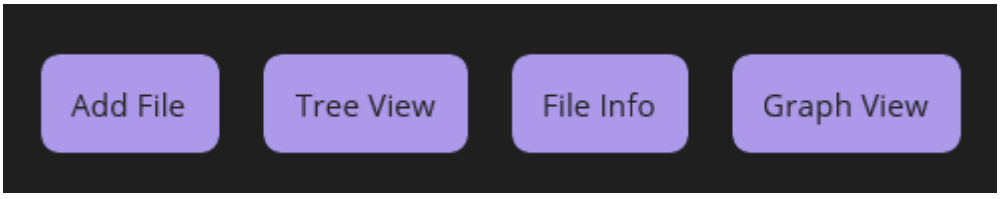


Рисунок 3.6 – Меню Додатку

Оскільки система передбачає можливість додавання нових нотаток до бази, користувачу потрібно мати відповідний інтерфейс який буде надавати

можливість додавати нові нотатки їх назви та текст до них, це реалізовано за допомогою простих полів для введеня тексту та кнопки в меню “Add File”, при натискані на дане меню користувач побачить прості поля з підказками що в якому з полів вводити, та кнопку для створення нової нотатки (рис. 3.7).

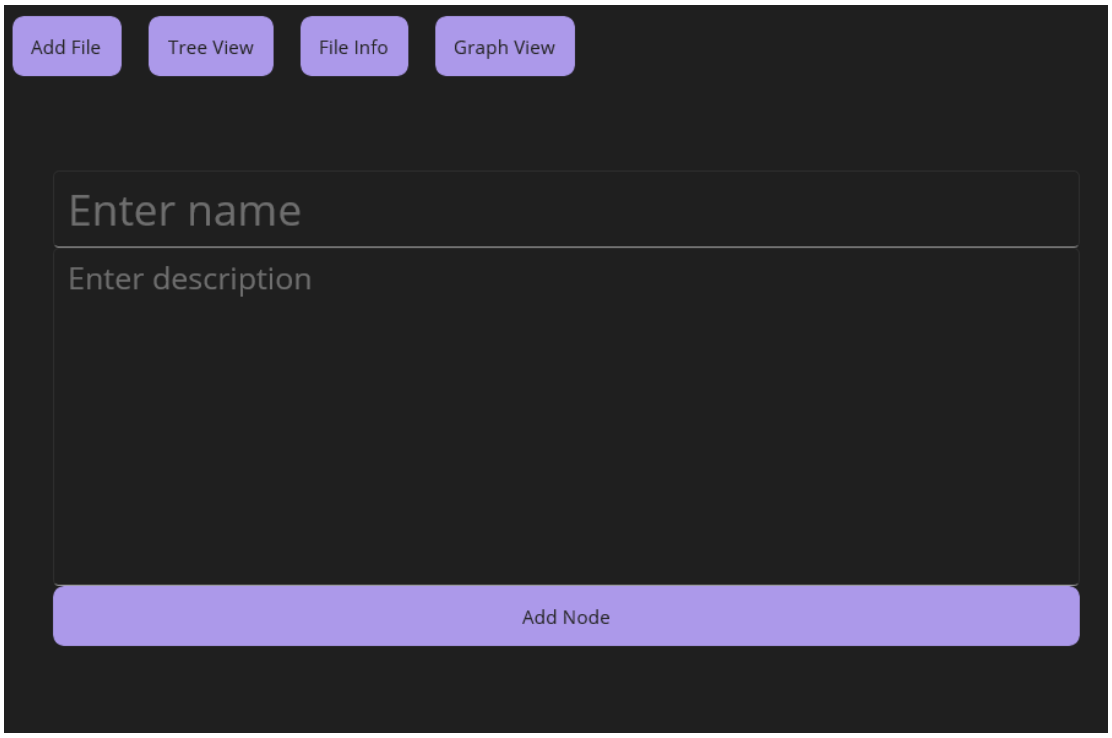


Рисунок 3.7 – Сторінка додавання нової нотатки.

Система також має можливість відображати нотатки у вигляді списку при натисканні на відповідну клавішу меню, користувач перейде до меню де будуть перераховані усі нотатки один за одним у вигляді списку (рис.3.8).

Даний інтерфейс надає користувачу легко знайти відповідну нотатку та бачити перелік створених нотаток в меню.

Рисунок 3.8 – Сторінка списку усіх нотаток

Система обов’язково має надавати можливість змінювати та видаляти існуючі нотатки, отже через що система надає можливість обрати нотатку в списку та видалити його або зміни його текст або назву, також завдяки можливості додавання посилань на інші нотатки користувача надається кнопка знайти посилання (рис. 3.9).

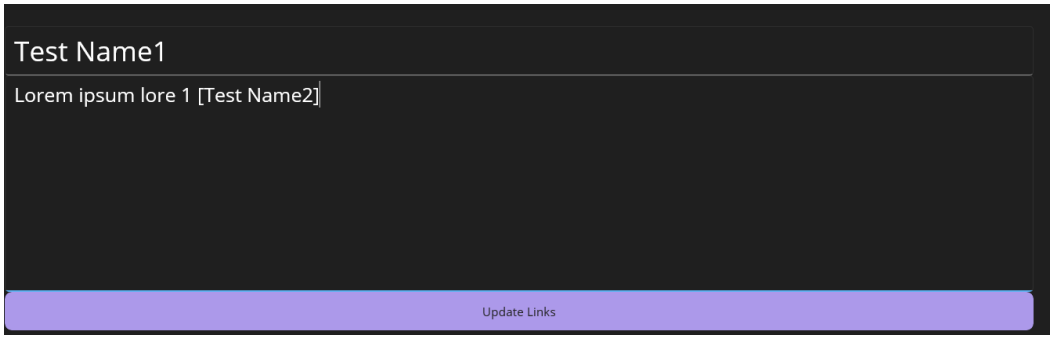


Рисунок 3.9 – Інтерфейс зміни нотатки

Також дана система надає користувачам візуалізувати їхні нотатки в інтерфейсі програмі у вигляді графу, дана система надає можливість користувачам подивитись на неї за допомогою графового вигляду (рис. 3.10), а саме кожна нотатка буде відображатись у вигляді вершини і кожен зв’язок у

вигляді грані, даний вигляд покращить можливість досліджувати зв’язки між нотатками та знаходити нові зв’язки та ідеї.

Розробка даного графічного інтерфейсу була проведена за допомогою бібліотеки .NET MAUI , а саме вбудованих функцій для відображення графічного інтерфейсу для кожної сторінки меню, та для кожної графічної яка була візуалізована.

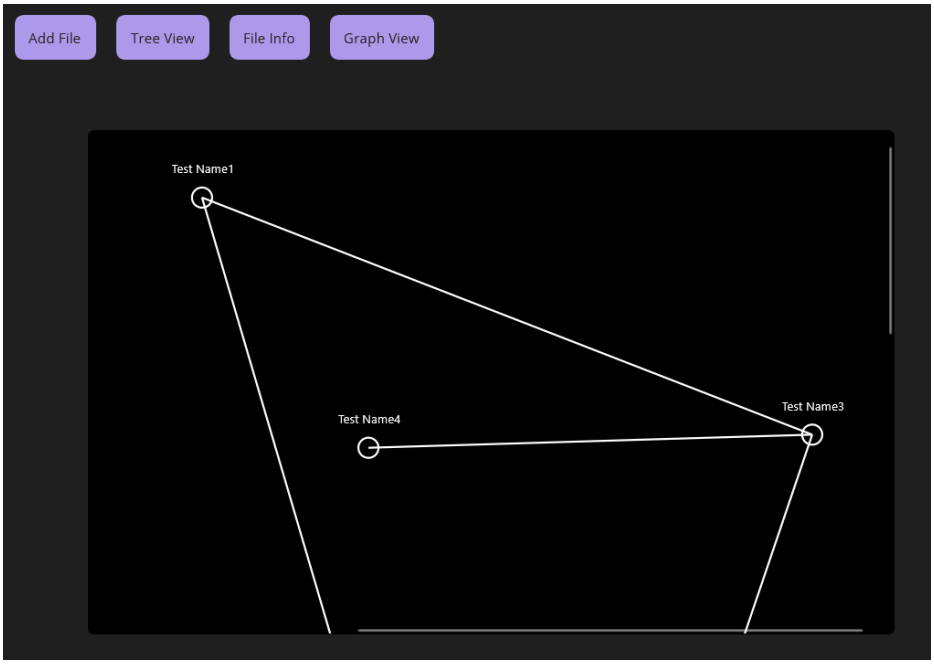


Рисунок 3.10 – Інтерфейс Графового вигляду

3.4 Розробка моделі системи

Для кращого розуміння процесу роботи програмних модулів мультиплатформеної системи, що базується на графах, було прийнято рішення розробити модель системи, за допомогою використання UML діаграми діяльності системи (рис. 3.11) [16].

Згідно з цією діаграмою, початковим етапом є вибір дії користувачем у програмі. Наприклад, якщо користувач хоче створити нову нотатку, він має після цього зберегти зміни. Якщо користувач бажає змінити існуючу нотатку, він

також має зберегти або відхилити зміни. Крім того, користувач може змінити обрану нотатку та переглянути його в формі списку або графу. Після виконання кожної дії, користувач має можливість завершити роботу з програмою або вибрати іншу функцію.

Ця діаграма, яка відображає послідовність дій користувача в системі, може бути у формі схеми або графічного інтерфейсу. Вона структурована таким чином, щоб зробити процес взаємодії користувача з системою якнайбільш зрозумілим та ефективним.

Починаючи з відкриття додатку або входу до системи, діаграма показує послідовність кроків, які користувач повинен виконати для досягнення своїх цілей або виконання конкретної функції. Ці кроки можуть включати введення даних, вибір опцій, натискання кнопок та перегляд реакції системи на кожну дію користувача.

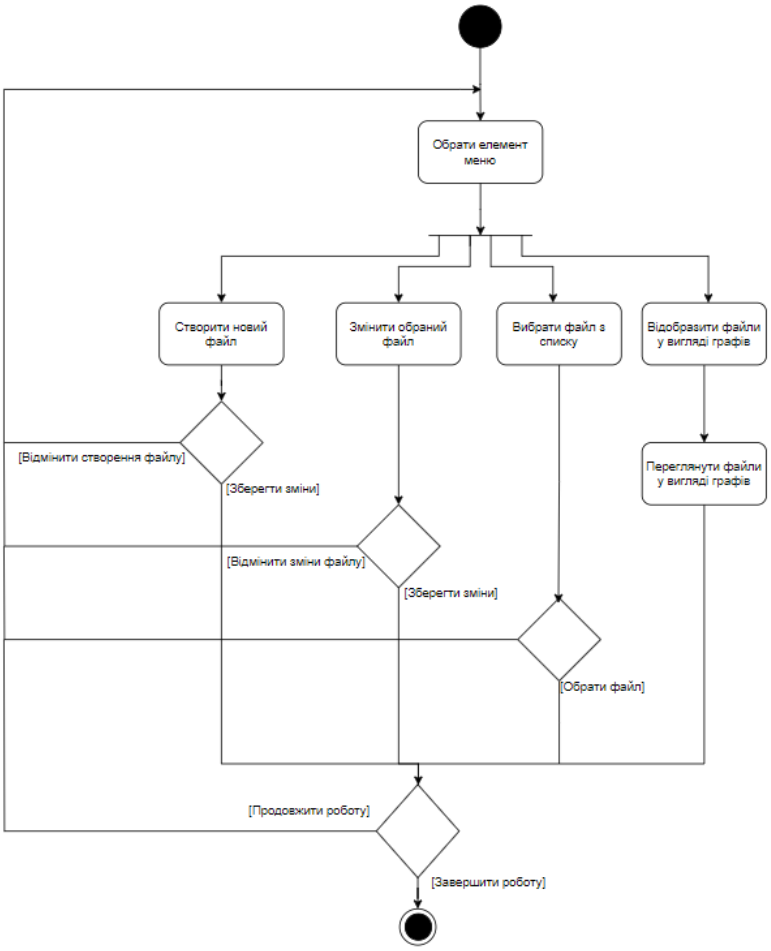


Рисунок 3.11 – Діаграма діяльності системи

Даний додаток буде поділений на 3 основних модулі які разом будуть формувати систему (рис. 3.12).

Першим важливим модулем є модуль збереження нотаток та їх зчитування з нотаток, він оперує з бібліотекою роботи з файлами та надає базові операції зберігання назв та тексту нотаток у вигляді файлів, також він надає можливість зчитати дані файли та видати у вигляді списку з структурами даних нотаток.

Другим модулем є модуль Графів, він має в собі загальний опис моделі графу, та його структури, також він має базові функції для роботи з графами.

Останнім важливим модулем є модуль Графічного відображення системи він являє собою головну частину яка поєднує та використовує два інші модулі для роботи з користувачем, збереженням його нотаток.

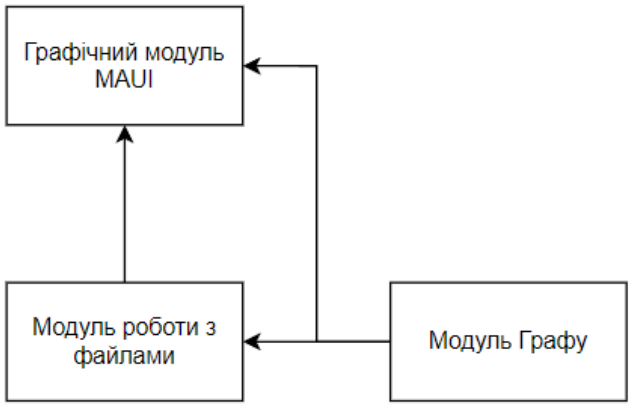


Рисунок 3.12 – Діаграма відношення модулів

3.5 Розробка алгоритмів роботи системи

Для розробки системи для зберігання нотаток на основі графів, необхідно зробити загальні алгоритми, згідно яких буде працювати додаток. Через великі обсяги додатку, і декілька модулів було створено ще й другу діаграму яка більш детально описує алгоритм відображення нотаток у вигляді графів.

Згідно першого алгоритму користувач спочатку отримує список усіх його нотаток, після чого він має можливість обрати один з них для зміни, після вибору користувача починається процес зміни нотатки.

Користувач отримує можливість змінювати інформацію в нотатці таку як ім’я та наповнення до поки не вирішить закінчити та натиснути на кнопку генерації нових посилань.

При завершені процесу генерації нових посилань користувач може повернутись до списку нотаток та повторити даний алгоритм, або перейти до меню системи та продовжити роботу там, дана дія завершує роботу наведеного.

Згідно другого алгоритму, а саме алгоритму відображення графу першим етапом є створення пустого списку граней, даний список буде

використовуватись далі в алгоритмі для створення унікальних граней та пропуску уже існуючих для покращення використання ресурсів системи, що допоможе зробити систему більш ефективною та швидкою.

Наступним етапом є вибір наступної вершини в списку, потім вибір зв’язку з списку зв’язків даної вершини після чого йде перевірка чи координату цього зв’язку існують, якщо ні то додаються, потім іде обробка наступного зв’язку поки не будуть обробленні всі зв’язки, після чого вешина відображується і починається повторне обиарння нової вершини, після закінчення всіх вершин алгоритм відображає всі зібрані зв’язки у вигляді ліній між вершинами, та завершує роботу (рис. 3.13).

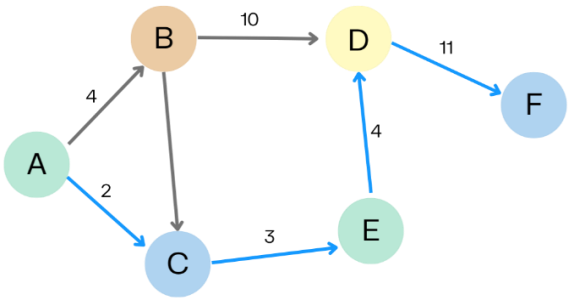


Рисунок 3.13 – Алгоритм роботи пошуку в графі.

Розглянувши дані алгоритми, можна зрозуміти, що розроблений застосунок дає змогу користувачам створювати та змінювати свої нотатки у вигляді нотаток та створювати легко зв’язки між ними.

Після цього користувачі можуть переглядати їх завдяки їх графовому представленню, яке дозволяє візуально відображати зв'язки між нотатками. Таке графове представлення полегшує сприйняття структури даних та допомагає користувачам швидше зорієнтуватися в своїй інформації.

Нижче наведено алгоритм, який використовуючи функції .NET MAUI, надає змогу користувачам бачити їхній алгоритм. Цей алгоритм використовує цикли для обробки кожної нотатки та створення списків зв’язків між ними.

Кожна нотатка розглядається окремо, а її зв’язки з іншими нотатками записуються у вигляді списку. Після цього інформація про зв’язки відображається один поруч з одним, створюючи граф, який відображає зв'язки між нотатками (рис. 3.14).

Цей підхід дозволяє ефективно відобразити структуру даних у вигляді графу, що допомагає користувачам краще зрозуміти взаємозв'язки між їхніми нотатками та легше навігувати по ним. Використання .NET MAUI надає можливість побудувати зручний та інтуїтивно зрозумілий інтерфейс для відображення цього графу користувачам у зручному форматі.

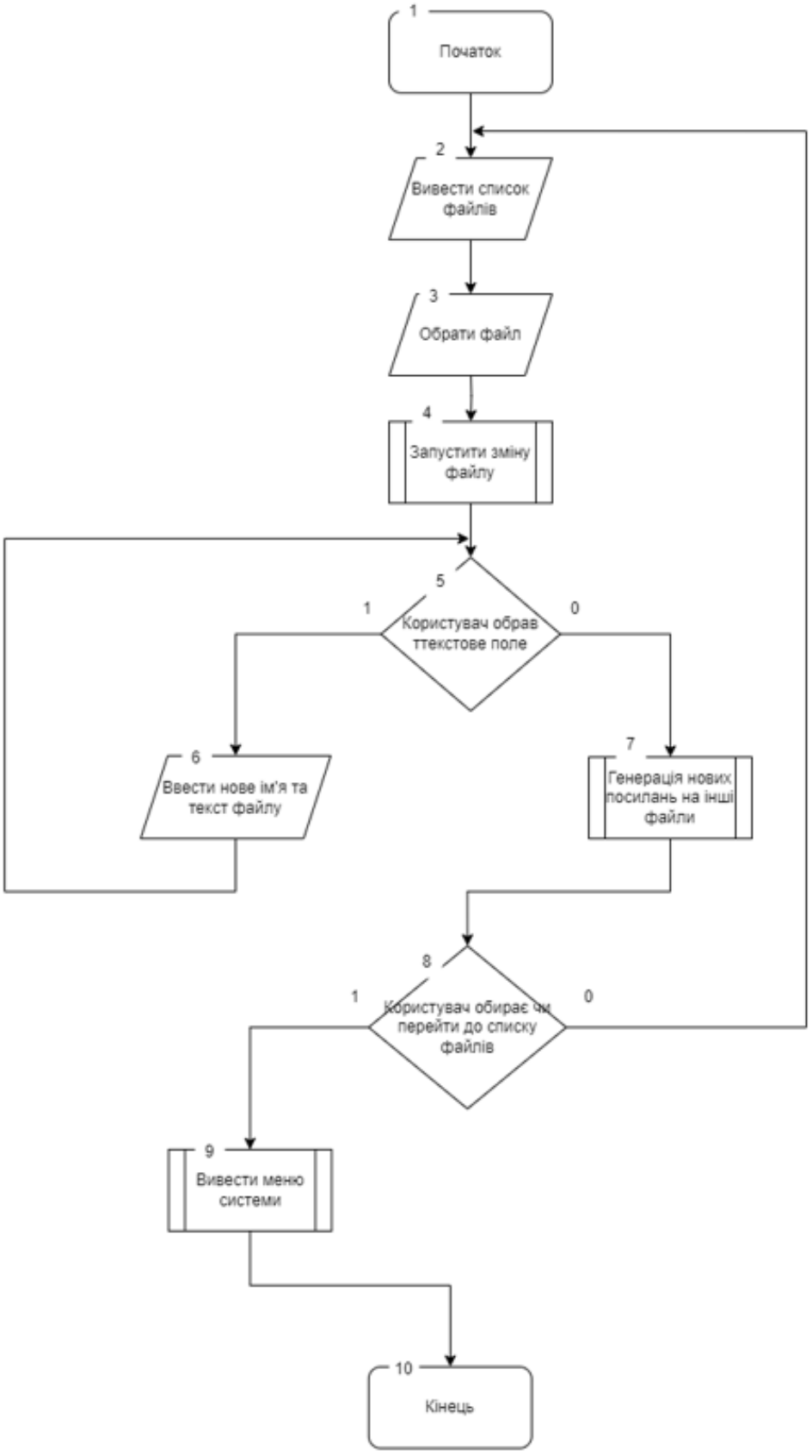


Рисунок 3.14 – Схема алгоритму для зміни нотатки

3.6 Розробка модуля відображення об’єктів в вигляді графів

Однією з головною задач системи є відображення нотаток у 2D графі [17]. Процес відображення графу у різних системах працює по різному, але завдяки використанню .NET MAUI можна зробити загальний код який буде використовувати Canvas для відображення частин графу [18], що в свою чергу наддасть можливість фреймворку самому відповідати за відображення на різних пристроях.

Для відображення графу, потрібно дізнатись та вирішити багато проблем [19], однією з них є розуміння що саме має бути частиною даного відображення (рис. 3.15).

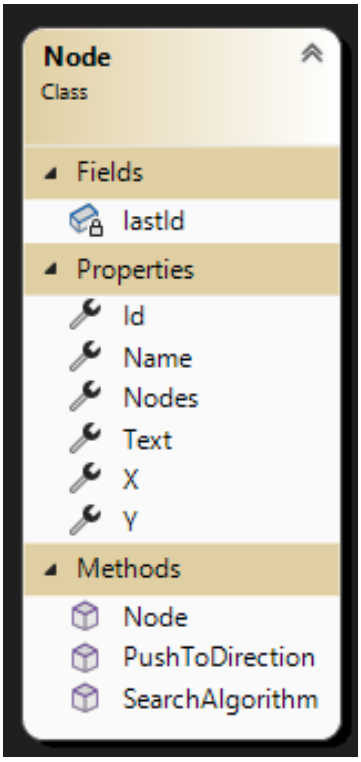


Рисунок 3.15 – Структура класу вершини

Структура класу вершини включає в себе декілька ключових атрибутів. На першому місці зазвичай знаходиться назва нотатки, яка ідентифікує вершину та є основним ідентифікатором. Далі, текст нотатки може бути включений у

структуру вершини для забезпечення доступу до вмісту нотатки безпосередньо з графової структури.

Посилання на інші вершини є ключовим аспектом структури графу і використовується для визначення зв'язків між вершинами. Це дозволяє моделювати взаємозв'язки та залежності між нотатками або об'єктами. Нарешті, координати вершини вказують її місцеположення у візуальному представленні графу. Це важливо для коректного відображення та навігації користувача в графічному інтерфейсі програми.

Як було описано раніше, Для кожної точки, вихідні координати не є ключовими; головне - їхні зв'язки. Це дозволяє генерувати випадкові координати для кожної точки на початку процесу відображення.

Після створення координат відбувається ітерація через всі точки та їх зв'язки. Потім для кожної точки відображається коло з її ім'ям, і на координатах відбувається візуалізації ліній, що представляють зв'язки між точками.

Після впровадження вказаного алгоритму для відображення графів користувач отримає докладне 2D графічне представлення своїх нотаток. Ця візуалізація структури та організації нотаток у формі графа може значно полегшити їх розуміння та сприйняття.

Користувачі зможуть краще розуміти зв'язки між нотатками, їх ієрархію та взаємозв'язки, що сприятиме більш ефективній навігації та управлінню ними. Графічне представлення дозволить швидше виявляти залежності між різними частинами проекту, розуміти логіку взаємодії та шляхи оптимізації.

Нижче наведено діаграму роботи алгоритму для відображення нотаток у графічному представлені (рис. 3.16).

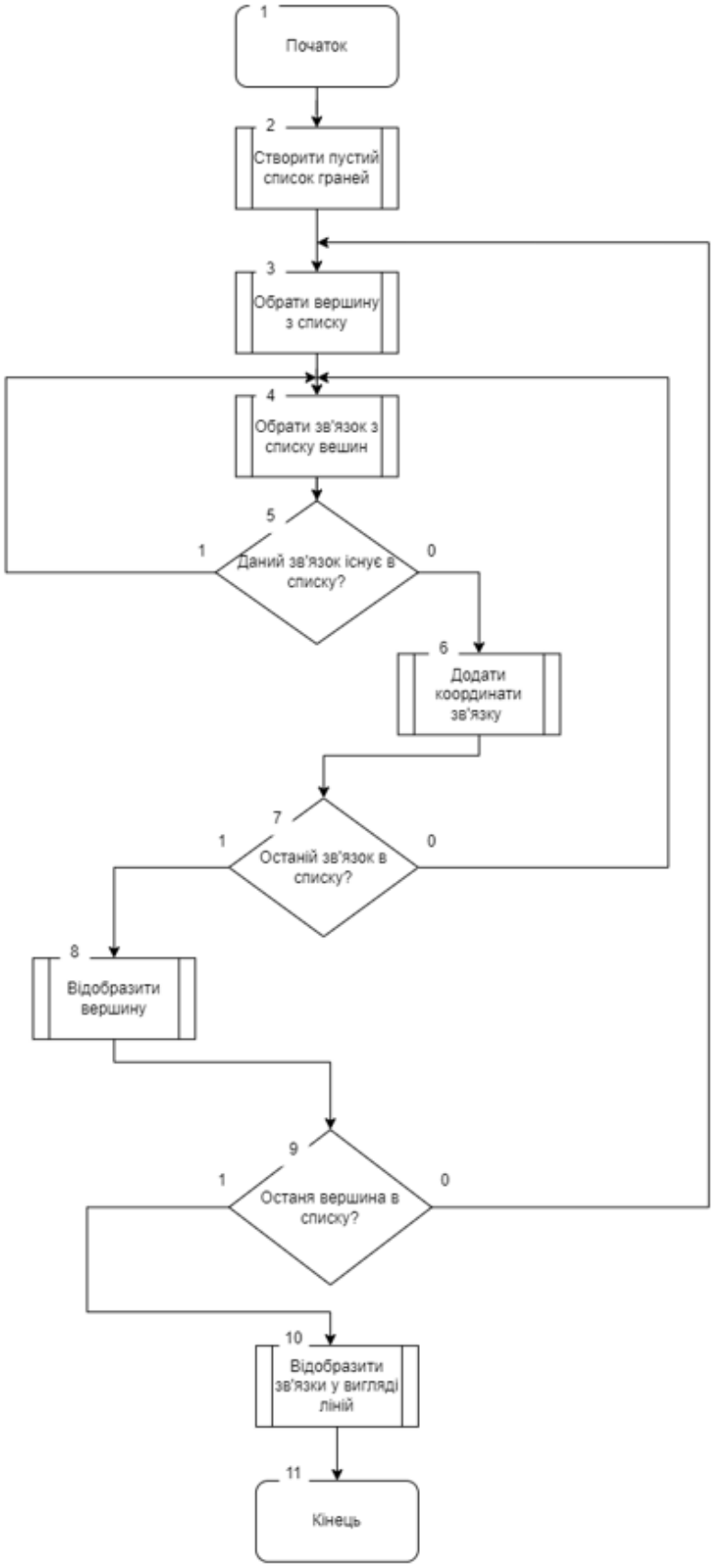


Рисунок 3.16 – Схема алгоритму для відображення графу

Користувачі зможуть краще розуміти зв'язки між нотатками, їх ієрархію та взаємозв'язки, що сприятиме більш ефективній навігації та управлінню ними. Графічне представлення дозволить швидше виявляти залежності між різними частинами проекту, розуміти логіку взаємодії та шляхи оптимізації.

Це суттєво підвищить продуктивність та зручність роботи з нотатками для користувачів програмного забезпечення (рис. 3.17).

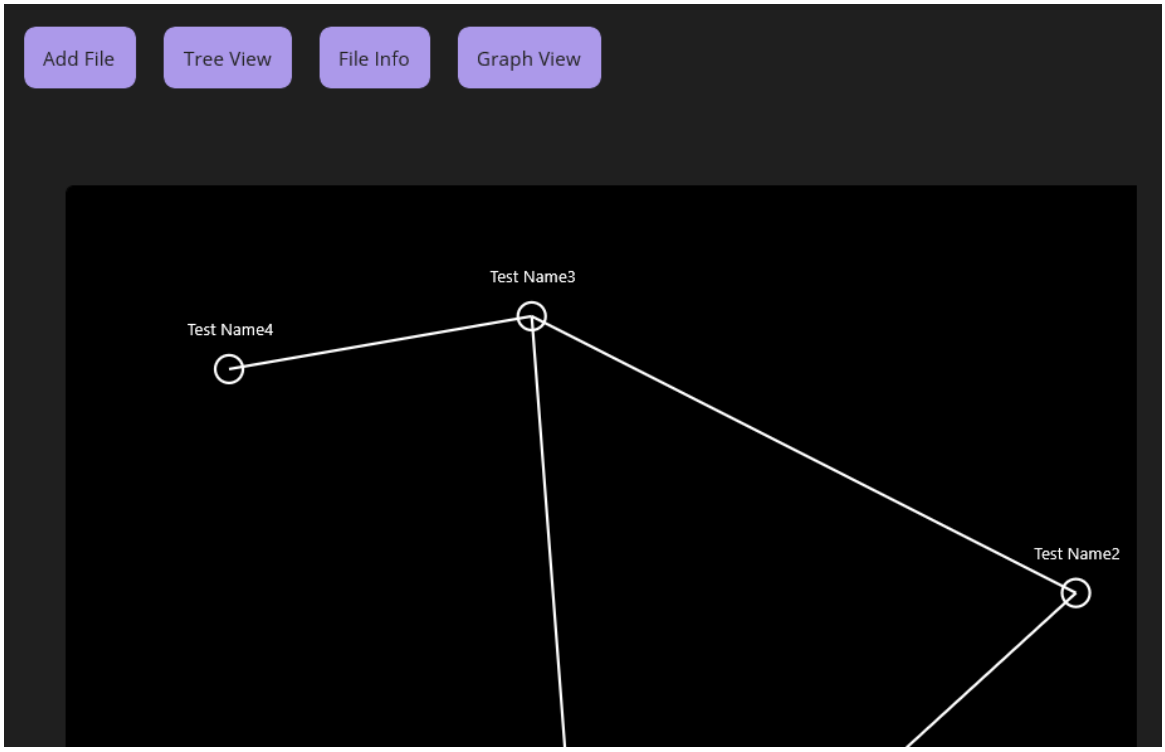


Рисунок 3.17 – Результат візуалізації графу

3.7 Розробка модуля зміни нотаток та додавання зв’язків

Основною задачею даного модуля є надання користувачу можливості змінювати його нотатки завдяки системі та після цього знаходити та додавати зв’язки. Для цього модуль має вбудовані інструменти для редагування тексту та назви нотатки, що забезпечує зручний та простий спосіб внесення змін. Крім того, система надає можливість здійснювати пошук нотаток за різними

критеріями та додавати до них зв’язки, які допомагають організувати та структурувати інформацію. Такий функціонал дозволяє користувачам ефективно працювати зі своїми нотатками та легко встановлювати зв’язки між ними для полегшення навігації та роботи з даними.

Через те що користувач працює з нотатками основними речами які він може змінити є назва та текст нотатки, виходячи з цього модуль повинен надавати можливість надавати різні поля та алгоритми для зміни цих полів, даний функціонал реалізовано за допомогою 2 типів полів вводу (рис. 3.18), які надає .NET MAUI.

```
Node currentNode = nodes.First(node=> node.Id == selectedNodeId);

Entry nameEntry = new Entry
{
    Placeholder = "Enter name",
    Text = currentNode.Name,
    FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Entry))
};

nameEntry.TextChanged += (sender, args) => {
    currentNode.Name = args.NewTextValue;
};

// Create and configure Editor for description
Editor descriptionEditor = new Editor
{
    Placeholder = "Enter description",
    Text = currentNode.Text,
    HeightRequest = 250,
    FontSize = Device.GetNamedSize(NamedSize.Medium, typeof(Editor))
};

descriptionEditor.TextChanged += (sender, args) => {
    currentNode.Text = args.NewTextValue;
};
```

Рисунок 3. 18 – Лістинг коду для створення 2 полів для вводу

Entry – є простим компонентом введення тексту, де користувач може вводити однорядковий текст (рис. 3.19). Він може бути використаний для отримання невеликих кількостей текстових даних від користувача, таких як ім'я, електронна пошта, пароль тощо.

Editor – є компонентом введення тексту для більш довгих текстових даних. Він зазвичай використовується для введення багаторядкового тексту, такого як

текстові коментарі, описи, повідомлення тощо. В Editor можна вводити великі обсяги тексту, і він зазвичай має можливості форматування тексту, такі як жирний, курсив, список тощо.

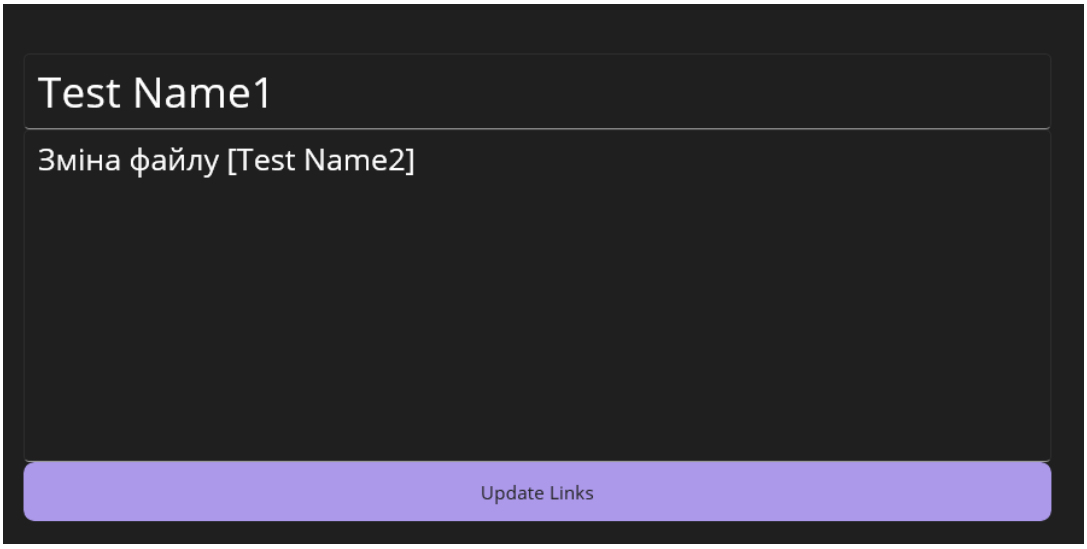


Рисунок 3. 19 – Графічний вигляд зміни нотаток

Для даного випадку компонент Entry буде використаний для імені нотатки, а компонент Editor для тексту, це дасть користувачу зручні та інтерактивні елементи інтерфейсу.

У даному модулі ключовим інструментом є кнопка, яка виконує важливу функцію створення посилань до інших нотаток на основі аналізу тексту нотаки. Ця кнопка використовує спеціально розроблений для цієї цілі алгоритм, який взаємодіє з вмістом нотатки та іншими нотатками у проекті.

Алгоритм отримує текст нотатки та аналізує його, виявляючи місця з квадратними дужками, які можуть вказувати на імена інших нотаток (рис. 3.19). Після цього алгоритм перевіряє наявність цих нотаток у проекті. Якщо вони знайдені, алгоритм додає нові посилання до списку посилань цієї нотатки, що допомагає створювати зв'язки між різними частинами проекту та полегшує навігацію користувача.

```
private List<Node> CheckForNodes(string text)
{
    var linkedNodes = new List<Node>();

    List<string> resultList = new List<string>();
    int startIndex = -1;

    for (int i = 0; i < text.Length; i++)
    {
        if (text[i] == '[')
        {
            startIndex = i + 1;
        }
        else if (text[i] == ']' && startIndex != -1)
        {
            string extractedText = text.Substring(startIndex, i - startIndex);
            resultList.Add(extractedText);
            startIndex = -1;
        }
    }

    foreach (var item in resultList)
    {
        var refNode = nodes.FirstOrDefault(node => node.Name.ToLower() == item.ToLower());
        if (refNode != null)
        {
            linkedNodes.Add(refNode);
        }
    }

    return linkedNodes;
}
```

Рисунок 3.20 – Лістинг коду алгоритму пошуку посилань по тексту

3.8 Висновки

У третьому розділі роботи був обґрунтований вибір мови програмування та технологій для розробки системи. Після детального аналізу потреб програмної системи та вимог до нього було прийнято рішення використовувати мову програмування C# та фреймворк .NET.

Обрано мову програмування C# через її широкі можливості, високу продуктивність та розповсюдженість в індустрії програмного забезпечення. Вона є однією з найпопулярніших мов для розробки програмного забезпечення на платформі .NET, що дозволяє забезпечити стабільність та ефективність роботи системи.

Також було проведено дослідження, було звернено увагу на ретельний аналіз як вхідних, так і вихідних даних програмної системи. Це дозволило отримати глибоке розуміння функціональності системи та виявити можливі напрями оптимізації. Зокрема, звернено увагу на потенційну можливість покращення продуктивності системи шляхом зберігання лише одного екземпляру зв’язків. Для досягнення цієї мети було розроблено та впроваджено

основний алгоритм заміни нотаток, який дозволяє ефективно керувати зв’язками та оптимізувати використання ресурсів системи.

Крім того, для забезпечення кращої візуалізації структури зв’язків у системі розроблено алгоритм відображення графу. Цей алгоритм дозволяє користувачам отримати зрозуміле та зручне візуальне представлення структури зв’язків, що допомагає полегшити розуміння функціональності програми та сприяє швидкому виявленню можливих проблем.

З метою детального опису функціональності та взаємодії компонентів системи також розроблено модель роботи за допомогою UML діаграм. Ця модель надає повний огляд взаємодії між різними компонентами системи, включаючи їхні функції та взаємозв’язки. Вона допомагає команді розробників та користувачам краще розібратися у функціональності системи та виявити можливі шляхи оптимізації та вдосконалення.

Крім того, для зручності розробки графічного інтерфейсу було обрано фреймворк MAUI. Це рішення було обрано через можливість написання графічного інтерфейсу з використанням мови C#, що спрощує розробку та підтримку системи. MAUI надає широкий набір інструментів для створення мультіплатформенних додатків, що відповідає вимогам сучасних розробок та дозволяє підтримувати систему на різних платформах без значних витрат часу та зусиль.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування мультиплатформеної системи потребує перевірки багатьох частин даної системи, зав’дяки використанню. MAUI можна розраховувати що система буде правильно працювати не залежно від платформи і буде достатньо протестувати тільки на одній платформі.

Для початку тестування необхідно запустити тестову версію програми, та провести відповідні функціональні тести які дозволять перевірити правильність роботи програми, дані тести включають тести на вивведеня та введеня даних.

Після проведення тестів на безпеку програми, включаючи тестування на проникнення, перевірку збереження конфіденційної інформації та аналіз вразливостей, використовуються спеціальні програмні засоби для моделювання потенційних атак та виявлення слабких місць. Ці засоби допомагають ідентифікувати можливі шляхи атак та вразливості системи, що можуть бути використані зловмисниками для отримання несанкціонованого доступу або порушення конфіденційності даних. Після виявлення потенційних проблем важливо вжити заходів для їх виправлення та підвищення рівня безпеки системи. Такий підхід дозволяє запобігти можливим загрозам та забезпечити надійний захист інформації користувачів.

Розробка системи — це складний і багатоетапний процес, що передбачає послідовну реалізацію кількох ключових етапів. Починаючи з аналізу вимог і визначення бізнес-цілей, розробники звертають увагу на всі аспекти проекту, включаючи функціональність, ергономіку, безпеку та ефективність. Після цього настає етап проектування, де створюються детальні схеми архітектури системи, вибираються технології та інструменти. Потім розробники переходять до фази реалізації, де кодують функціональність системи та інтегрують компоненти. Після завершення програмування настає етап тестування, під час якого

перевіряється працездатність, надійність та відповідність системи вимогам. Нарешті, після успішного завершення тестування відбувається етап впровадження системи, що включає у себе встановлення та налаштування програмного забезпечення, підготовку персоналу та підтримку в перші дні роботи. Кожен з цих етапів відіграє важливу роль у створенні успішної та ефективної системи, а їх послідовність та якість виконання визначають кінцевий результат проекту (рис. 4.1).

Рисунок 4.1 – Етапи розробки системи.

Розробка програмного забезпечення включає різноманітні види тестування, що виконуються на різних етапах процесу розробки. Функціональне тестування спрямоване на перевірку функціональних можливостей програми, тоді як нефункціональне тестування оцінює аспекти, такі як продуктивність та безпека. Модульне тестування перевіряє окремі компоненти програми, в той час як інтеграційне тестування оцінює взаємодію між цими компонентами. Системне тестування здійснюється на рівні всієї системи, перевіряючи її відповідність специфікації.

Регресійне тестування відстежує вплив нових змін на існуючу функціональність, а приймальне тестування проводиться після завершення усіх інших видів тестування для підтвердження готовності до випуску (рис. 4.2).

Рисунок 4.2 – Загальна карта видів тестування.

Мануальне та автоматизоване тестування - це дві ключові стратегії перевірки якості програмного забезпечення, кожна з яких має свої переваги та обмеження. Мануальне тестування виконується людськими тестувальниками, які вручну перевіряють функціональність системи, спостерігаючи за її реакцією та взаємодією з користувачем (рис. 4.3). Цей підхід дозволяє виявляти складні проблеми, що важко автоматизувати, та забезпечує більшу гнучкість у тестуванні нового функціоналу або внесених змін. З іншого боку, автоматизоване тестування використовує спеціальні програмні засоби для автоматизації процесу тестування, що дозволяє швидше виконувати повторювані тестові сценарії та зменшує ризик людських помилок. Цей підхід ефективний для регресійного тестування, коли потрібно перевірити, чи не

вплинули оновлення на вже існуючу функціональність. Враховуючи взаємодію обох підходів, команди розробників можуть досягти балансу між швидкістю та глибиною тестування, забезпечуючи високу якість системи.

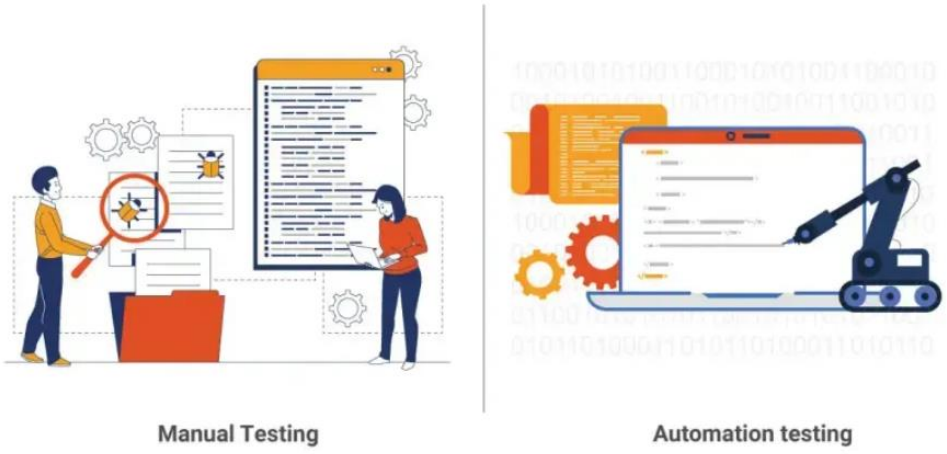


Рисунок 4.3 – Мануальне та автоматизоване тестування.

Після успішного завершення тестування на одній платформі, необхідно розширити його на інші платформи, що є важливим етапом в перевірці мультиплатформенності системи. MAUI допомагає автоматизувати цей процес, забезпечуючи можливість виконання тестів на різних платформах з використанням однієї і тієї ж самої тестової бази. Це дозволяє ефективно виявити та виправити будь-які платформено-залежні проблеми, забезпечуючи стабільну та надійну роботу системи на різних пристроях і операційних системах. Такий підхід зменшує час, необхідний для тестування, та гарантує високу якість системи перед випуском на ринок.

Крім того, після проходження базових функціональних тестів, важливо також звернути увагу на тести, пов'язані з безпекою програми. Вони включають в себе не лише перевірку вразливостей, а й аналіз стійкості до можливих атак. Застосування спеціальних програмних засобів дозволяє імітувати різні сценарії

атак та визначити слабкі місця в системі. Це допомагає підвищити рівень захисту програми та забезпечити конфіденційність та цілісність даних користувачів.

Додатково, для забезпечення надійності і оптимальної продуктивності системи, слід провести тестування на масштабування. Це дозволить визначити, як система веде себе при збільшенні обсягу даних та навантаження на різних платформах. Такий аналіз допоможе ідентифікувати можливі проблеми з продуктивністю та оптимізувати їх для забезпечення ефективної роботи системи навіть при великих навантаженнях.

Не менш важливим етапом тестування є перевірка сумісності програми з різними версіями операційних систем та пристроїв. Це включає в себе тестування на різних версіях Windows, macOS, Linux, iOS та Android, а також на різних моделях пристроїв з різними характеристиками. Такий підхід допомагає впевнитися, що програма працює коректно на широкому спектрі пристроїв і операційних систем, забезпечуючи задоволення користувачів незалежно від їхніх вподобань і потреб.

Після успішного завершення цих тестів проводиться оцінка продуктивності програми, включаючи тести на швидкодію, ефективне використання ресурсів мобільного пристрою та аналіз життєвого циклу програми. Результати тестування фіксуються у відповідній документації й використовуються для подальшого вдосконалення програми та забезпечення її якості на мобільних платформах, зокрема на платформі Android.

Результат тестування зображено на рис. 4.4.

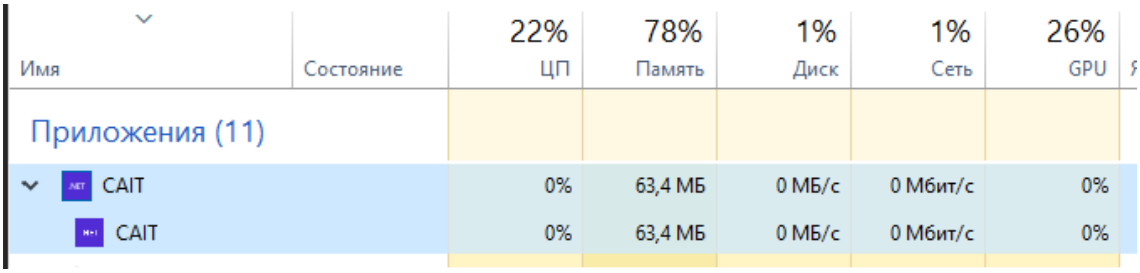


Рисунок 4.4 – Тестування продуктивності системи.

Також було проведено тестування використання процесора в режимі відлагодження, зокрема [20], оцінювалася його ефективність та стабільність під час виконання відлагоджувальних операцій (рис. 4.5).

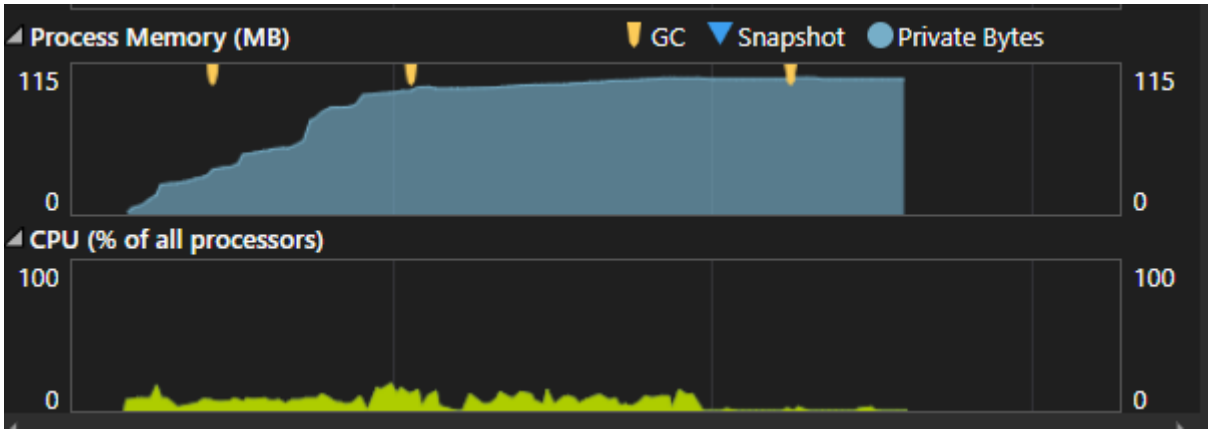


Рисунок 4.5 – Тестування продуктивності процесора системи.

Наступним кроком буде тестування одного із модулів системи а саме створення нотаток, для цього потрібно запустити застосунок та спробувати створити нову нотатку (рис. 4.6).

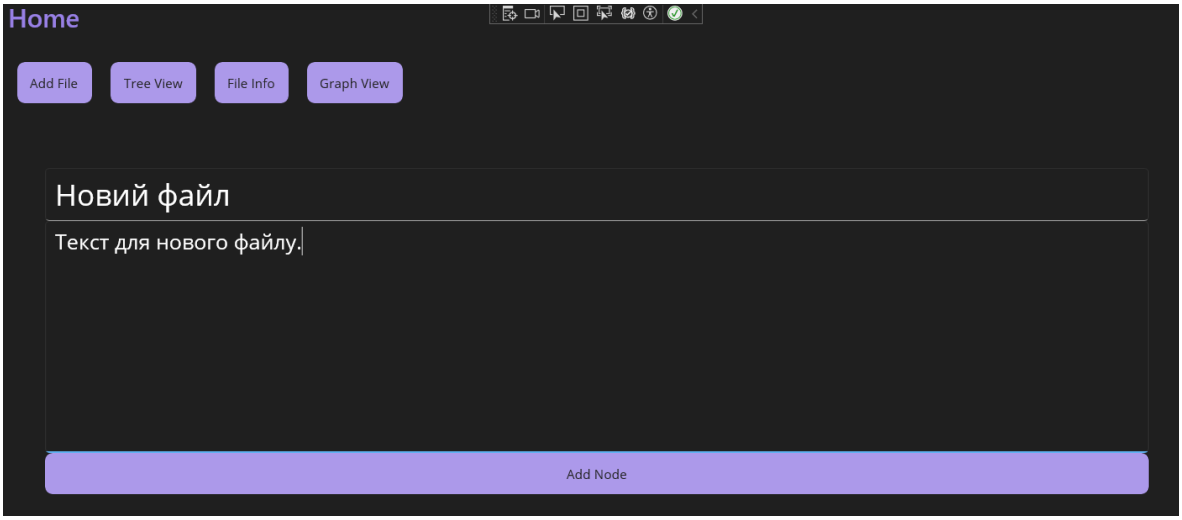


Рисунок 4.6 – Тестування додавання нової нотаки.

Також потрібно протестувати модуль який відповідає за відображення усіх існуючих нотаток користувача за допомогою списку, це є важливим модулем і

його можна протестувати якраз після додавання нової нотатки та перевірки чи є вона присутня в списку (рис. 4.7).

Рисунок 4.7 – Тестування відображення списку.

Також потрібно протестувати можливість користувача вибирати нотатки з списку дивитись текст нотатки, та мати можливість змінювати нотатку та додавати до неї посиланням до інших нотаток за допомогою відповідної кнопки (рис. 4.8).

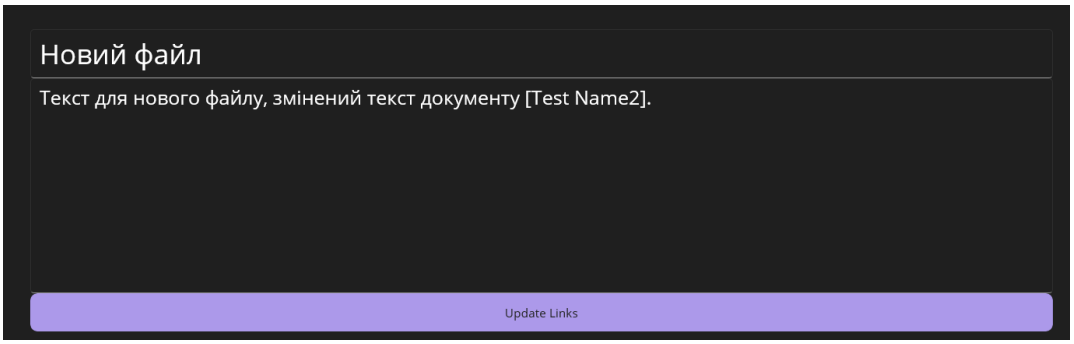


Рисунок 4.8– Тестування зміни нотатки.

І останньою частиною програми яку потрібно протестувати буде можливість перегляду графу у графічному режимі, а саме відображення всіх нотаток у вигляді вершин та граней, також має бути протестована можливість пересування по частинам графу за допомогою бокових ліній прокрутки (рис. 4.9).

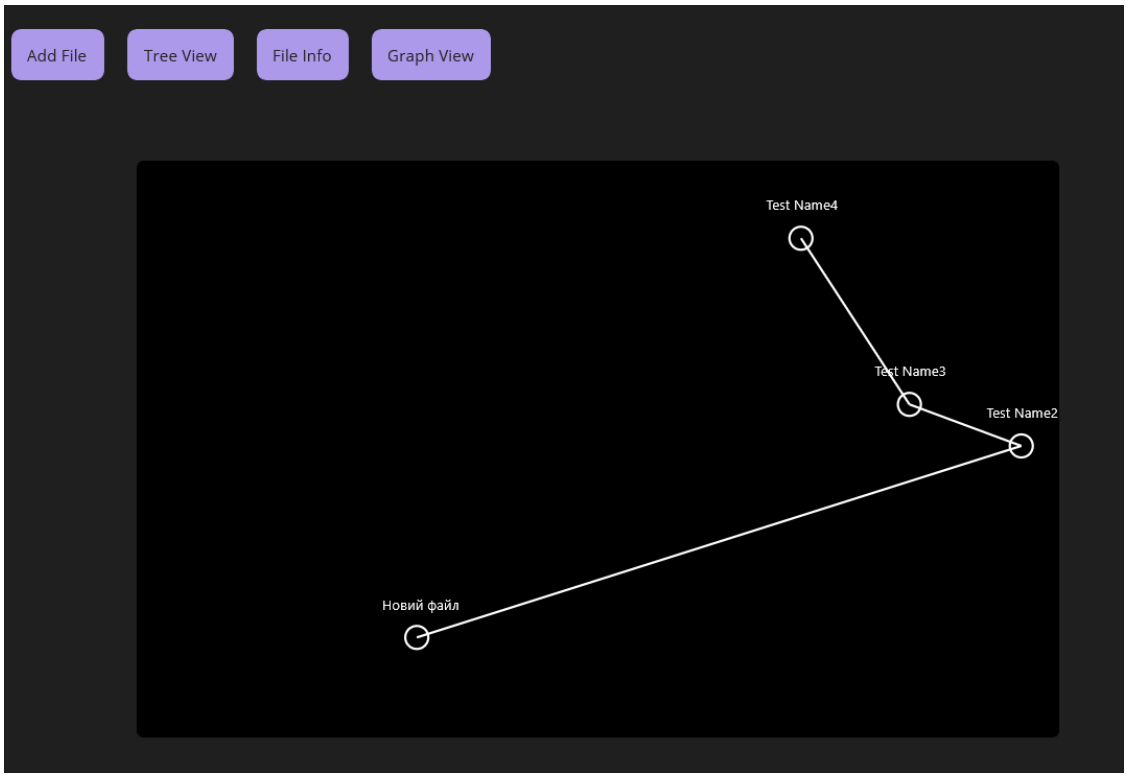


Рисунок 4.9– Тестування графового відображення

Також було проведено тестування і порівнняння алгоритмів пошуку, а саме звичайного лінійного алгоритму пошуку та пошуку за допомогою сортування по ступенні графу, в результаті стало зрозуміло що даний алгоритм краще працює з популярними вершинами які мають більше зв’язків.

Якщо відбувається пошук не популярного алгоритму, результати та близькі до лінійного, та стають гіршими. Результати тестування наведено в таблиці 4.1.

Швидкість вимірювалась за допомогою спеціальної бібліотеки .NET, вимірювання відбувались в тактах, також швидкість вимірювалась при 1000 вершин.

В результаті тестування стає очевидним, що даний алгоритм краще працює з популярними вершинами, але менш ефективний у випадку з не популярними, це свідчить про те, що популярні вершини мають більше зв'язків і, отже, алгоритм легше знаходить оптимальний шлях з ними.

Таблиця 4.1 – Результат тестування пошуку.

Умови тестування	Лінійний пошук	Пошук з сортуванням по степенях
Популярною вершиною	75	15
Середньо популярною вершиною	83	25
Мало популярною вершиною	78	60
Не популярною вершиною	80	110

З іншого боку, з не популярними вершинами алгоритм зазнає труднощів у виявленні оптимального шляху, оскільки вони можуть мати обмежену кількість зв'язків або ж навіть бути ізольованими від інших частин графа, в такому випадку лінійний алгоритм виграє по швидкості.

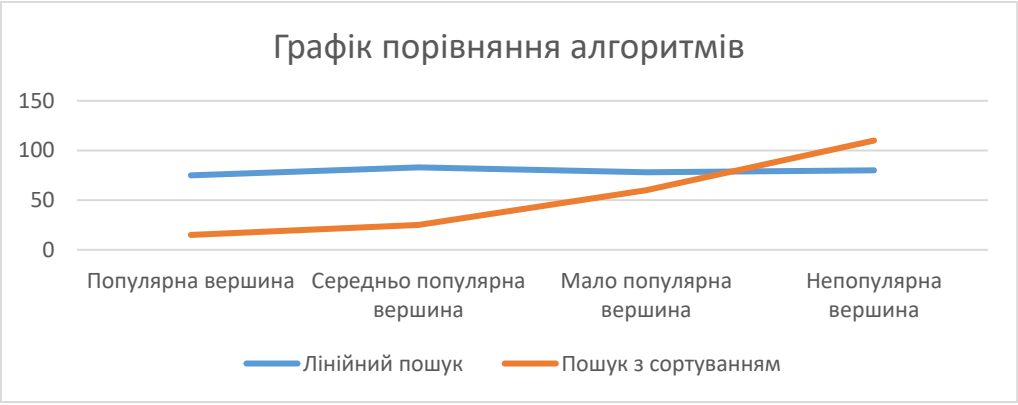


Рисунок 4.10 – Графік порівняння алгоритмів

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску системи [21]. Для забезпечення оптимальної роботи програми рекомендується мати достатньо вільного місця на пристрої, щоб уникнути перешкод у виконанні функцій і оптимізувати швидкодію. Також, рекомендується використовувати оновлену версію операційної системи Android для запобігання сумісності та безпекових проблем.

Важливо також мати на увазі, що деякі функції можуть бути недоступні на пристроях зі застарілими версіями операційних систем або з обмеженими технічними можливостями. Детальну інформацію щодо сумісності та оптимального використання системи можна знайти у розділі "Технічна підтримка" на офіційному веб-сайті програми.

Деталі щодо мінімальної та рекомендованої конфігурації для пристроїв Android можна знайти в таблиці 4.2.

Таблиця 4.2 – Мінімальна конфігурація для Android:

Процесор	ARM Cortex-A7
Оперативна пам'ять	512 МБ
Версія Android	Android 5.0 (Lollipop)
Простір зберігання	4 ГБ

Також нижче наведено рекомендовані характеристики для операційної системи Android, було визначено рекомендовану оперативну пам’ять для підтримки роботи програми, також було визначено рекомендований процесор, версію та загальне місце на системі, дані наведено на таблиці 4.3.

Таблиця 4.3 – Рекомендована конфігурація для Android:

Процесор	ARM Cortex-A53
Оперативна пам'ять	1 ГБ

Версія Android	Android 8.0 (Oreo)
Простір зберігання	4 ГБ

Також системи підтримує роботу на операційній системі iOS, нижче наведено мінімальні конфігурації для iOS на таблиці 4.4.

Таблиця 4.4 – Мінімальна конфігурація для iOS (iPhone та iPad):

Процесор	Apple A10
Оперативна пам'ять	1 ГБ
Версія iOS	iOS 12.0
Простір зберігання	1 ГБ

Також є ще і рекомендовані конфігурації саме для операційної системи iOS також підтримується і конфігурації вище ніж наведені, наведено на таблиці 4.5

Таблиця 4.5 – Рекомендована конфігурація для iOS (iPhone та iPad):

Процесор	Apple A11
Оперативна пам'ять	2 ГБ
Версія iOS	iOS 14.0
Простір зберігання	1 ГБ

Мультиплатформена система є надзвичайно гнучкою, оскільки вона може працювати на різних типах пристроїв, включаючи мобільні пристрої та комп’ютери.

Комп’ютери, на відміну від мобільних пристроїв, часто вимагають більше потужності та ресурсів для ефективної роботи. Це обумовлено не лише більшою фізичною розмірністю комп’ютерних систем, але й їх більшими можливостями та функціональністю.

Нижче представлені мінімальні конфігурації для операційних систем на базі Linux, MacOS та Windows на таблиці 4.6.

Таблиця 4.6 – Мінімальна конфігурація для Linux, MacOS та Windows:

Процесор	Intel Core i3-7100U (для Windows та Linux) / Intel Core i5-7360U (для macOS)
Оперативна пам'ять	4 ГБ
Версія ОС	Windows 10 / Linux Kernel 5 / macOS 10.12
Простір зберігання	2 ГБ

Також нижче наведені рекомендовані конфігурації, отримані в результаті тестування та перевірки конфігурацій, на сайтах для розробки програм на бібліотеці MAUI для комп’ютерних платформ таких як Linux, MacOS та Windows, вони потребують більш потужних апаратних вимог та досі не є занадто високими для типового користувача, конфігурації наведені на таблиці 4.7.

Таблиця 4.7 – Рекомендована конфігурація для Linux, MacOS та Windows:

Процесор	Intel Core i7-11800H (для Windows та Linux) / Apple M1 Pro (для macOS)
Оперативна пам'ять	8 ГБ
Версія ОС	Windows 11 / Linux Kernel 5.10 / macOS 12
Простір зберігання	2 ГБ

Після запуску системи користувач може починати роботу, спочатку йому потрібно створити декілька різних нотаток за допомогою інструментів створення нотаток, після чого користувач може почати їх змінювати за допомогою

системних інструментів, або додавати до них зв’язки та змінювати відповідні нотатки.

При наявності 2 та більше нотаток користувач може створити граф з зв’язками даних нотаток за допомогою відповідної клавіші меню, після чого користувач може досліджувати зв’язки між його нотатками.

Користувач має можливість також переглядати свої нотатки у вигляді списку для легшого пошуку потрібної нотатки, після того як користувач найде відповідну нотатку користувач може обрати та подивитись його деталі або змінити його.

Користувач має можливість змінювати усі його нотатки за допомогою, відповідного інтерфейсу він може змінювати назву та текст нотатки, після цих змін користувач має можливість натиснути відповідну клавішу та знайти зв’язки в тексті до інших нотаток.

4.3 Висновки

У четвертому розділі було проведено тестування програмних модулів та системи в цілому, було перевірено скільки процесор потребує системних ресурсів та інші показники, продуктивності та ефективності програми. Було проведено тестування та порівняння лінійного алгоритму пошуку, з власним розробленим алгоритмом, в результаті чого було отримано графік продуктивності алгоритму в різних ситуаціях. Також було розроблено мінімальні та рекомендовані конфігурації для платформ на які може бути встановлена система. Було розроблено інструкцію користувача по тому, як почати роботу з програмою та використовувати її функції.

ВИСНОВКИ

У бакалаврській дипломній роботі було розроблено систему ведення нотаток зі збереженням даних з використанням графів. Для розробки було використано середовище програмної розробки Visual Studio. Бакалаврську дипломну роботу виконано згідно методичних вказівок [22].

Було проаналізовано стан проблеми на сьогоднішній день. Розглянуто аналоги системи, та створено порівняльну характеристику для аналогів та власної системи. Базуючись на проведених дослідженнях було вирішено, що розробка є доцільною та сформульовано основні цілі, задачі розробки.

Запропоновано формальний опис процесу нотування даних із застосуванням апарату теорії графів. Вдосконалено метод обходу графу в глибину, що дозволило розширити сферу застосування DFS на графи, що містять кілька деревовидних структур або ізольовані вершини. Вдосконалено метод візуалізації графу, що дозволило отримати більш точну та зрозумілу візуалізацію структури графу та полегшує загальне розуміння взаємозв'язків вершин.

В результаті аналізу технологій для розробки мультиплатформеної системи було визначено варіанти реалізації й обґрунтовано вибір мови програмування та фреймворків та бібліотек для створення системи. Було обрано C# як мову програмування, .NET як головний фреймворк на основі якого має бути побудована система та обрано бібліотеку MAUI для створення мультиплатформеного застосунку для роботи з системою.

Було розроблено схеми та структури для загального алгоритму, та інших модулів системи за допомогою UML діаграм.

Тестування програми довело повну працездатність даної системи та відповідність технічному завданню. Також було розроблено технічні конфігурації для системи на різних пристроях, та інструкцію користувача. Таким чином, мету бакалаврської дипломної роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Graph storage technique. Medium : веб-сайт. URL: https://medium.com/@siddarthsiddhu58/graph-storage-technique-7d32861956f0 (дата звернення: 02.03.2024).

2. Graph Data Structure And Algorithms. Geeks for geeks : веб-сайт. URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/ (дата звернення: 05.03.2024).

3. Грицина В. В., Ткаченко О. М. Мультиплатформена система для нотування та зберігання даних на основі графів. Матеріали Всеукраїнська науково-технічна конференція підрозділів Вінницького національного технічного університету (ВНТКП ВНТУ), м.Вінниця, 20-22 березня 2024 р. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2024/paper/view/20503/16972

4. Visualizations of Graph Algorithms. Algorithms discrete mathematics : веб-сайт. URL: https://algorithms.discrete.ma.tum.de/ (дата звернення: 15.03.2024).

5. Roam Research a note-taking tool for networked thought. Roam research: веб-сайт. URL: https://roamresearch.com/ (дата звернення: 23.03.2024).

6. Workflowy Organize Your Brain. Workflowy : веб-сайт. URL: https://workflowy.com/ (дата звернення: 02.04.2024).

7. Notion . Notion : веб-сайт. URL: https://www.notion.so/ (дата звернення: 02.04.2024).

8. Evernote tame your work, organize your life . Evernote : веб-сайт. URL: https://evernote.com/ (дата звернення: 04.04.2024).

9. Depth First Search or DFS for a Graph . Geek for geeks : веб-сайт. URL: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/ (дата звернення: 06.04.2024).

10. Breadth First Search(BFS) in C#. Dot net for all : веб-сайт. URL: https://www.dotnetforall.com/breadth-first-searchbfs-in-c/ (дата звернення: 07.04.2024).

11. C# language documentation. Learn Microsoft : веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 11.04.2024).

12. .NET Microsoft cross platform framework . Dot net Microsoft : веб-сайт. URL: https://dotnet.microsoft.com/en-us/ (дата звернення: 12.04.2024).

13. MAUI Multi-platform App UI. Dot net Microsoft : веб-сайт. URL: https://dotnet.microsoft.com/en-us/apps/maui (дата звернення: 13.04.2024).

14. Visual Studio Integrated development environment. Visual Studio Microsoft : веб-сайт. URL: https://visualstudio.microsoft.com/en/#vs-section (дата звернення: 14.04.2024).

15. MAUI User interface. Learn Microsoft : веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/controls/?view=net-maui-8.0 (дата звернення: 16.04.2024).

16. What is Unified Modeling Language (UML). Visual paradigm : веб-сайт. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/#:~:text=UML%2C%20short%20for%20Unified%20Modeling,business%20modeling%20and%20other%20non%2D (дата звернення: 18.04.2024).

17. Two-dimensional graph. Hand wiki : веб-сайт. URL: https://handwiki.org/wiki/Two-dimensional_graph (дата звернення: 20.04.2024).

18. Draw Graphical objects in MAUI. Learn Microsoft : веб-сайт. URL: https://learn.microsoft.com/en-us/dotnet/maui/user-interface/graphics/draw?view=net-maui-8.0 (дата звернення: 21.04.2024).

19. Graph theory in Discrete Mathematics. Java Point : веб-сайт. URL: https://www.javatpoint.com/graph-theory-in-discrete-mathematics (дата звернення: 02.05.2024).

20. Measure memory usage in Visual Studio. Learn Microsoft : веб-сайт. URL: https://learn.microsoft.com/en-us/visualstudio/profiling/memory-usage?view=vs-2022 (дата звернення: 05.05.2024).

21. Створіть посібник користувача. IT Pedia : веб-сайт. URL: https://uk.itpedia.nl/2018/06/25/een-gebruikershandleiding-maken/ (дата звернення: 06.05.2024).

22. Романюк О. Н., Чорноволик Г. О., Методичні вказівки до виконання бакалаврських дипломних робіт для студентів спеціальності 121 «Інженерія програмного забезпечення»: методичні – вказівки Вінниця : ВНТУ, 2022. 51 с.

ДОДАТКИ

Додаток А – Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

" 12 " березня 2024 р.

Технічне завдання
на бакалаврську дипломну роботу
«Розробка мультиплатформеної системи
для нотування та зберігання даних на основі графів»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:

к.т.н., доц. каф. ПЗ. О.М.Ткаченко

" 12 " березня 2024 р.

Виконав:

студент гр.1ПІ-20б В.В.Грицина

" 12 " березня 2024 р.

Вінниця – 2024 року

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка мультиплатформеної системи для нотування та зберігання даних на основі графів».

Галузь застосування – системи нотування.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ 11 березня 2024 р. №80 ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є розширення функціональних можливостей ведення нотаток та перегляду їх зв’язків за рахунок застосування теорії графів, що дозволить користувачам систематизувати дані і створить передумови для отримання нових знань на їх основі.

Призначення роботи – зберігання та візуалізація нотаток для студентів, освітян та дослідників.

4 Вихідні дані для проведення БДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Graph storage technique. Medium : веб-сайт. URL: https://medium.com/@siddarthsiddhu58/graph-storage-technique-7d32861956f0
2. Graph Data Structure And Algorithms. Geeks for geeks : веб-сайт. URL: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
3. Visualizations of Graph Algorithms. Algorithms discrete mathematics : веб-сайт. URL: https://algorithms.discrete.ma.tum.de/
4. Depth First Search or DFS for a Graph . Geek for geeks : веб-сайт. URL: https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/

5. Технічні вимоги

Базові методи заповнення тексту, методи створення файлових нотаток на основі теорії графів, методи пошуку по графу Depth-first search та Breadth-first search, максимальна кількість тексту в файлах не обмежена, дані та координати про розташування вершин та граней у результуючому графів, кінцеве зображення мапи нотаток та їх зв’язків.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред’являється по закінченню робіт:

– пояснювальна записка до БДР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз розвитку систем нотування	14.03.24 - 31.03.24
2	Розробка методів нотування на основі теорії графів	01.04.24 - 18.04.24
3	Розробка алгоритмів роботи з нотатками та інтерфейсу системи	19.04.24 - 06.05.24
4	Тестування системи	07.05.24 - 24.05.24
5	Оформлення матеріалів до захисту БДР	25.05.24 - 30.05.24

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської дипломної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б – Протокол перевірки роботи

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка мультиплатформеної системи для нотування та зберігання даних на основі графів

Тип роботи: Бакалаврська дипломна робота

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доцент кафедри ПЗ Ткаченко О.М

Оригінальність	93.8
Схожість	6.2

Аналіз звіту подібності

▪ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи Грицина В.В.

Керівник роботи Ткаченко О.М.

Додаток В – Лістинг програми

```
using NodeApi;

namespace CAIT;

public class GraphDrawable : IDrawable
{
    private List<Node> _nodes;

    public GraphDrawable(List<Node> nodes)
    {
        _nodes = nodes;
    }

    public void Draw(ICanvas canvas, RectF dirtyRect)
    {
        Random rnd = new Random();
        canvas.StrokeSize = 2;
        canvas.StrokeColor = Colors.White;
        canvas.FontColor = Colors.White;

        HashSet<LinkCord> links = new HashSet<LinkCord>();

        //Set Random Cords
        foreach (Node node in _nodes)
        {
            node.X = rnd.Next(10, (int)dirtyRect.Width);
            node.Y = rnd.Next(10, (int)dirtyRect.Height);
        }

        //Simulate Physics

        const int iterations = 15;
```

```
            for (int i = 0; i < iterations; i++)
            {
                foreach (var node in _nodes)
                {

                    //Push to linked nodes
                    foreach (var linkedNode in node.Nodes)
                    {
                        if( VectorDistance(node.X, node.Y, linkedNode.X, linkedNode.Y) < 15)
                        {
                            node.PushToDirection(CalculateDirection(node.X,  node.Y,  linkedNode.X,
linkedNode.Y));
                        }
                    }

                    // Push out of linked nodes
                    var closeNodes = _nodes
                        .Where(cnode => VectorDistance(node.X, node.Y, cnode.X, cnode.Y) < 10f)
                        .ToList();

                    int[] directions = {0,0,0,0};

                    closeNodes.ForEach(cnode =>
                    {
                        var dir = CalculateDirection(node.X, node.Y, cnode.X, cnode.Y);
                        directions[(int)dir]++;
                    });

                    int max = directions[0];
                    Direction maxDir = Direction.Left;
                    for (int d = 1; d < 4; d++)
                    {
                        if (directions[d] > max)
```

```
            {
                max = directions[d];
                maxDir = (Direction)d;
            }
        }

        node.PushToDirection(maxDir);
    }
}

//Draw Circles
foreach (Node node in _nodes)
{
    foreach (var childNode in node.Nodes)
    {
        links.Add(new LinkCord()
        {
            startPosX = node.X,
            startPosY = node.Y,
            endPosX = childNode.X,
            endPosY = childNode.Y
        });
    }

    canvas.DrawCircle(node.X, node.Y, 10);
    canvas.DrawString(node.Name, node.X - 30, node.Y - 25, HorizontalAlignment.Left);
}

//Draw Lines
foreach (var link in links)
{
    canvas.DrawLine(link.startPosX, link.startPosY, link.endPosX, link.endPosY);
}
}
```

```
private static float VectorDistance(float x1, float y1, float x2, float y2)
{
    return (float)Math.Sqrt(Math.Pow(x2 - x1, 2) + Math.Pow(y2 - y1, 2));
}

private static Direction CalculateDirection(float x1, float y1, float x2, float y2)
{
    float dx = x2 - x1;
    float dy = y2 - y1;

    double angle = Math.Atan2(dy, dx) * (180 / Math.PI);

    if (angle < 0)
    {
        angle += 360;
    }

    if (angle >= 45 && angle < 135)
    {
        return Direction.Up;
    }
    else if (angle >= 135 && angle < 225)
    {
        return Direction.Left;
    }
    else if (angle >= 225 && angle < 315)
    {
        return Direction.Down;
    }
    else
    {
        return Direction.Right;
    }
}
```

```
}

<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
             x:Class="CAIT.MainPage">

    <StackLayout>
        <HorizontalStackLayout Margin="10">
            <Button Margin="10" x:Name="MenuButton1" Text="Add File"
Clicked="MenuItem_Clicked" />
            <Button Margin="10" x:Name="MenuButton2" Text="Tree View"
Clicked="MenuItem_Clicked" />
            <Button Margin="10" x:Name="MenuButton3" Text="File Info"
Clicked="MenuItem_Clicked" />
            <Button Margin="10" x:Name="MenuButton4" Text="Graph View"
Clicked="MenuItem_Clicked" />
        </HorizontalStackLayout>

        <!-- Container for dynamically changing content -->
        <ContentView Margin="50" x:Name="ContentContainer">
            <StackLayout>
                <Image Margin="10" WidthRequest="300" HeightRequest="300"
Source="cait_logo.png" />
                <Label Margin="5" Text="Graph Notes" HorizontalTextAlignment="Center"/>
            </StackLayout>
        </ContentView>
    </StackLayout>

</ContentPage>

<?xml version="1.0" encoding="UTF-8" ?>
<Shell
    x:Class="CAIT.AppShell"
```

```
    xmlns="http://schemas.microsoft.com/dotnet/2021/maui"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:CAIT"
    Shell.FlyoutBehavior="Disabled"
    Title="CAIT">

    <ShellContent
        Title="Home"
        ContentTemplate="{DataTemplate local:MainPage}"
        Route="MainPage" />

</Shell>

using System.Text.Json;
using Azure.Storage.Files.Shares;
using NodeApi;

namespace FileStorageAPI
{
    public class FileStorage
    {
        private string azureStorageConnectionString = "your_connection_string";
        private string shareName = "your_share_name";

        public void SaveAllNodes(List<Node> nodes, string folderPath)
        {
            // Serialize the list of nodes to JSON
            string json = JsonSerializer.Serialize(nodes);

            // Write JSON to a file
            string filePath = Path.Combine(folderPath, "nodes.json");
            File.WriteAllText(filePath, json);

            // Upload file to Azure file storage
```

```
            ShareClient share = new ShareClient(azureStorageConnectionString, shareName);
            ShareDirectoryClient directory = share.GetDirectoryClient(folderPath);
            ShareFileClient file = directory.CreateFile("nodes.json");

            using (FileStream stream = File.OpenRead(filePath))
            {
                file.Upload(stream);
            }
        }

        public List<Node> ReadAllNodes(string folderPath)
        {
            // Download file from Azure file storage
            ShareClient share = new ShareClient(azureStorageConnectionString, shareName);
            ShareDirectoryClient directory = share.GetDirectoryClient(folderPath);
            ShareFileClient file = directory.GetFileClient("nodes.json");

            MemoryStream memoryStream = new MemoryStream();
            file.DownloadTo(memoryStream);
            memoryStream.Position = 0;

            // Deserialize JSON to list of nodes
            using (StreamReader reader = new StreamReader(memoryStream))
            {
                string json = reader.ReadToEnd();
                return JsonSerializer.Deserialize<List<Node>>(json);
            }
        }
    }
}

namespace NodeApi;
public enum Direction
{
```

```
    Left,
    Right,
    Up,
    Down
}


namespace NodeApi
{
    public class LinkCord
    {
        public float startPosX;
        public float startPosY;
        public float endPosX;
        public float endPosY;

        public override bool Equals(object obj)
        {
            if (obj == null || GetType() != obj.GetType())
                return false;

            LinkCord other = (LinkCord)obj;
            return startPosX == other.startPosX &&
                   startPosY == other.startPosY &&
                   endPosX == other.endPosX &&
                   endPosY == other.endPosY;
        }
        public override int GetHashCode()
        {
            unchecked
            {
                int hash = 17;
                hash = hash * 23 + startPosX.GetHashCode();
                hash = hash * 23 + startPosY.GetHashCode();
```

```
            hash = hash * 23 + endPosX.GetHashCode();
            hash = hash * 23 + endPosY.GetHashCode();
            return hash;
        }
    }

  }
}


namespace NodeApi
{
  public class Node
  {
    private static int lastId = 1;

    public Node()
    {
      Id = lastId++;
      Nodes = new List<Node>();
    }

    public void PushToDirection(Direction dir)
    {
      float distance = 10;
      switch (dir)
      {
        case Direction.Left:
          X += distance;
          break;
        case Direction.Right:
          Y -= distance;
          break;
        case Direction.Up:
```

```
                Y += distance;
                break;
            case Direction.Down:
                X -= distance;
                break;
            default:
                throw new ArgumentException("Invalid direction provided.");
        }
    }

    public static Node? SearchAlgorithm(List<Node> nodes, string nameToSearch)
    {
        return nodes.OrderByDescending(node => node.Nodes.Count).FirstOrDefault(node=> node.Name.Contains(nameToSearch));
    }

    public int Id { get;}
    public string Name { get; set; }
    public string Text { get; set; }
    public float X { get; set; }
    public float Y { get; set; }
    public List<Node> Nodes { get; set; }
  }
}
using Microsoft.Extensions.Logging;

namespace CAIT
{
  public static class MauiProgram
  {
    public static MauiApp CreateMauiApp()
    {
      var builder = MauiApp.CreateBuilder();
      builder
```

```
            .UseMauiApp<App>()
            .ConfigureFonts(fonts =>
            {
                fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");
                fonts.AddFont("OpenSans-Semibold.ttf", "OpenSansSemibold");
            });

#if DEBUG
                builder.Logging.AddDebug();
#endif

        return builder.Build();
    }
  }
}

using NodeApi;
using System.Text.RegularExpressions;

namespace CAIT
{
  public partial class MainPage : ContentPage
  {
    private List<Node> nodes;
    private int selectedNodeId = 1;
    public MainPage()
    {
      InitializeComponent();

      nodes = RGenerator.GenerateNodes(1000);
    }

    private void MenuItem_Clicked(object sender, EventArgs e)
    {
```

```
    var menuItem = (Button)sender;

    // Depending on the selected menu item, set the content of the ContentView
    switch (menuItem.Text)
    {
        case "Add File":
            showAddFile();
            break;
        case "Tree View":
            showTree();
            break;
        case "File Info":
            showFileInfo();
            break;
        case "Graph View":
            showGraphView();
            break;
    }
}

private void showAddFile()
{
    Entry nameEntry = new Entry
    {
        Placeholder = "Enter name",
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Entry))
    };

    // Create and configure Editor for description
    Editor descriptionEditor = new Editor
    {
        Placeholder = "Enter description",
        HeightRequest = 250,
        FontSize = Device.GetNamedSize(NamedSize.Medium, typeof(Editor))
```

```
    };

    Button AddNode = new Button()
    {
        Text = "Add Node"
    };

    AddNode.Clicked += (sender, args) =>
    {
        var newNode = new Node()
        {
            Name = nameEntry.Text,
            Text = descriptionEditor.Text
        };
        newNode.Nodes = CheckForNodes(newNode.Text);
        nodes.Add(newNode);
    };

    //Add elements to container
    ContentContainer.Content = new StackLayout
    {
        Children =
        {
            nameEntry,
            descriptionEditor,
            AddNode
        }
    };
}

private void showTree()
{
    var stack = new StackLayout();
```

```
//Add search button
Entry search = new Entry()
{
   Placeholder = "Search By Name",
   FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Entry))
};

Button searchButton = new Button
{
   Text = "Search"
};

searchButton.Clicked += (sender, args) =>
{
   var filter = search.Text;
   if (!string.IsNullOrEmpty(filter))
   {

      var watch1 = System.Diagnostics.Stopwatch.StartNew();
      var node = Node.SearchAlgorithm(nodes, filter);
      watch1.Stop();
      var elapsedMs1 = watch1.ElapsedMilliseconds;

      var watch2 = System.Diagnostics.Stopwatch.StartNew();
      node = nodes.FirstOrDefault(x => x.Name.Contains(filter));
      watch2.Stop();
      var elapsedMs2 = watch2.ElapsedMilliseconds;

      stack.Children.Clear();
      stack.Children.Add(search);
      stack.Children.Add(searchButton);
      if (node != null)
      {
         drawTree(stack, [node]);
```

```
            }
            else
            {
                drawTree(stack, nodes);
            }
        }
        else
        {
            stack.Children.Clear();
            stack.Children.Add(search);
            stack.Children.Add(searchButton);
            drawTree(stack, nodes);
        }
    };

    //Adding each node
    stack.Children.Add(search);
    stack.Children.Add(searchButton);
    drawTree(stack, nodes);

    //Add elements to container

    ContentContainer.Content = new ScrollView()
    {
        Content = stack,
        VerticalScrollBarVisibility = ScrollBarVisibility.Always,
        HeightRequest = 500,
        WidthRequest = 800,
        Orientation = ScrollOrientation.Vertical
    };
}

private void drawTree(StackLayout stack, List<Node> nodes)
{
```

```
    foreach (var node in nodes)
    {
        var nodeLink = new Button() { Text = node.Name, Margin = 10 };
        nodeLink.Clicked += (sender, args) =>
        {
            selectedNodeId = node.Id;
            showFileInfo();
        };
        stack.Children.Add(nodeLink);
    }
}

private void showFileInfo()
{
    Node currentNode = nodes.First(node=> node.Id == selectedNodeId);

    Entry nameEntry = new Entry
    {
        Placeholder = "Enter name",
        Text = currentNode.Name,
        FontSize = Device.GetNamedSize(NamedSize.Large, typeof(Entry))
    };

    nameEntry.TextChanged += (sender, args) => {
        currentNode.Name = args.NewTextValue;
    };

    // Create and configure Editor for description
    Editor descriptionEditor = new Editor
    {
        Placeholder = "Enter description",
        Text = currentNode.Text,
        HeightRequest = 240,
        FontSize = Device.GetNamedSize(NamedSize.Medium, typeof(Editor))
```

```
};

descriptionEditor.TextChanged += (sender, args) => {
    currentNode.Text = args.NewTextValue;
};

Button updateLinksButton = new Button
{
    Text = "Update Links"
};

updateLinksButton.Clicked += (sender, args) =>
{
    currentNode.Nodes = CheckForNodes(descriptionEditor.Text);
};

Button deleteButton = new Button
{
    Text = "Delete"
};

deleteButton.Clicked += (sender, args) =>
{
    nodes.Remove(currentNode);
    showTree();
};

//Add elements to container
ContentContainer.Content = new StackLayout
{
    Children =
    {
        nameEntry,
        descriptionEditor,
```

```
            updateLinksButton,
            deleteButton
        }
    };
}

private void showGraphView()
{
    int maxSizeXY = 800;

    var graphicsView = new GraphicsView
    {
        HeightRequest = maxSizeXY,
        WidthRequest = maxSizeXY,
        BackgroundColor = Colors.Yellow,
        Drawable = new GraphDrawable(nodes)
    };

    var scrollView = new ScrollView
    {
        Content = graphicsView,
        HorizontalScrollBarVisibility = ScrollBarVisibility.Always,
        VerticalScrollBarVisibility = ScrollBarVisibility.Always,
        HeightRequest = 500,
        WidthRequest = 800,
        Orientation = ScrollOrientation.Both
    };

    var frame = new Frame
    {
        BorderColor = Colors.Black, // Set the color of the border
        CornerRadius = 5, // Set the corner radius of the border (optional)
        HeightRequest = 500,
        WidthRequest = 800,
```

```
            Content = scrollView
        };



        ContentContainer.Content = frame;
    }

    private List<Node> CheckForNodes(string text)
    {
        var linkedNodes = new List<Node>();

        List<string> resultList = new List<string>();
        int startIndex = -1;

        for (int i = 0; i < text.Length; i++)
        {
            if (text[i] == '[')
            {
                startIndex = i + 1;
            }
            else if (text[i] == ']' && startIndex != -1)
            {
                string extractedText = text.Substring(startIndex, i - startIndex);
                resultList.Add(extractedText);
                startIndex = -1;
            }
        }

        foreach (var item in resultList)
        {
            var refNode = nodes.FirstOrDefault(node => node.Name.ToLower() ==
item.ToLower());
            if (refNode != null)
```

```
            {
                linkedNodes.Add(refNode);
            }
        }

        return linkedNodes;
    }
  }
}
```

Додаток Г – Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

Розробка мультиплатформеної системи
для нотування та зберігання даних на основі графів

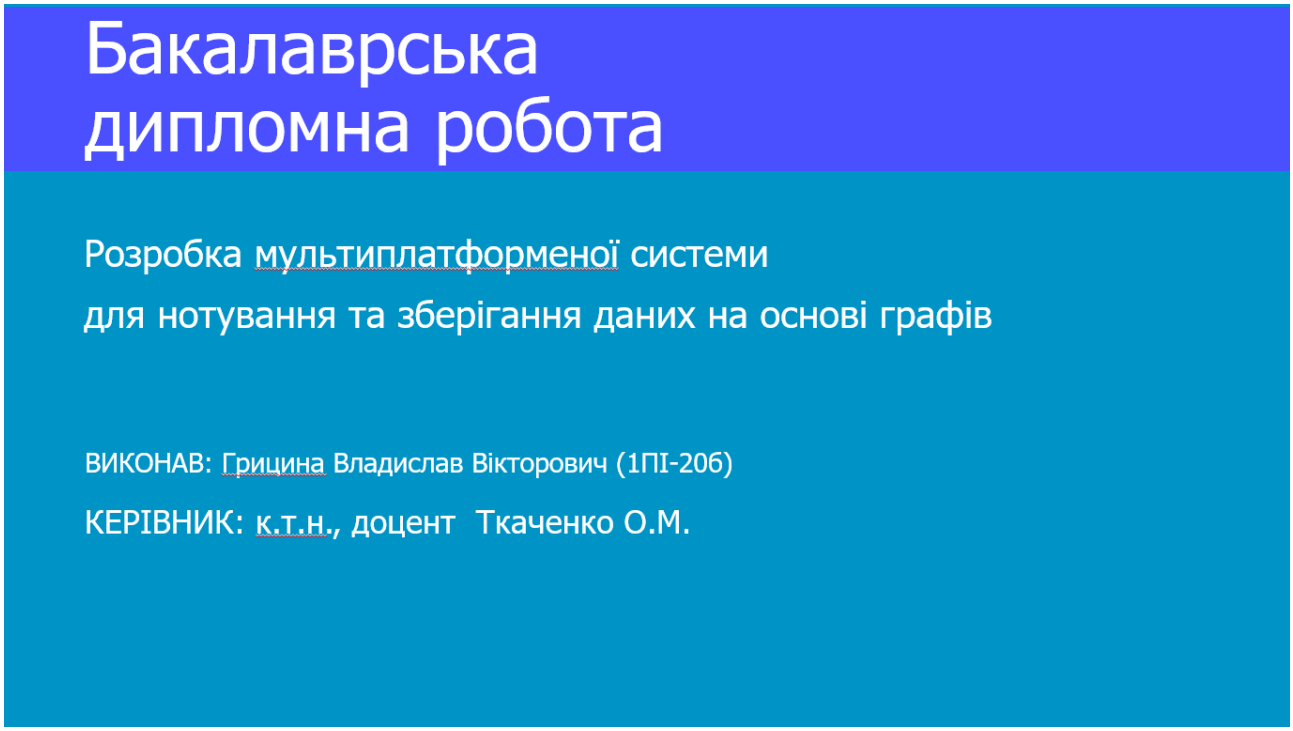


Рисунок Г.1 – Слайд презентації №1

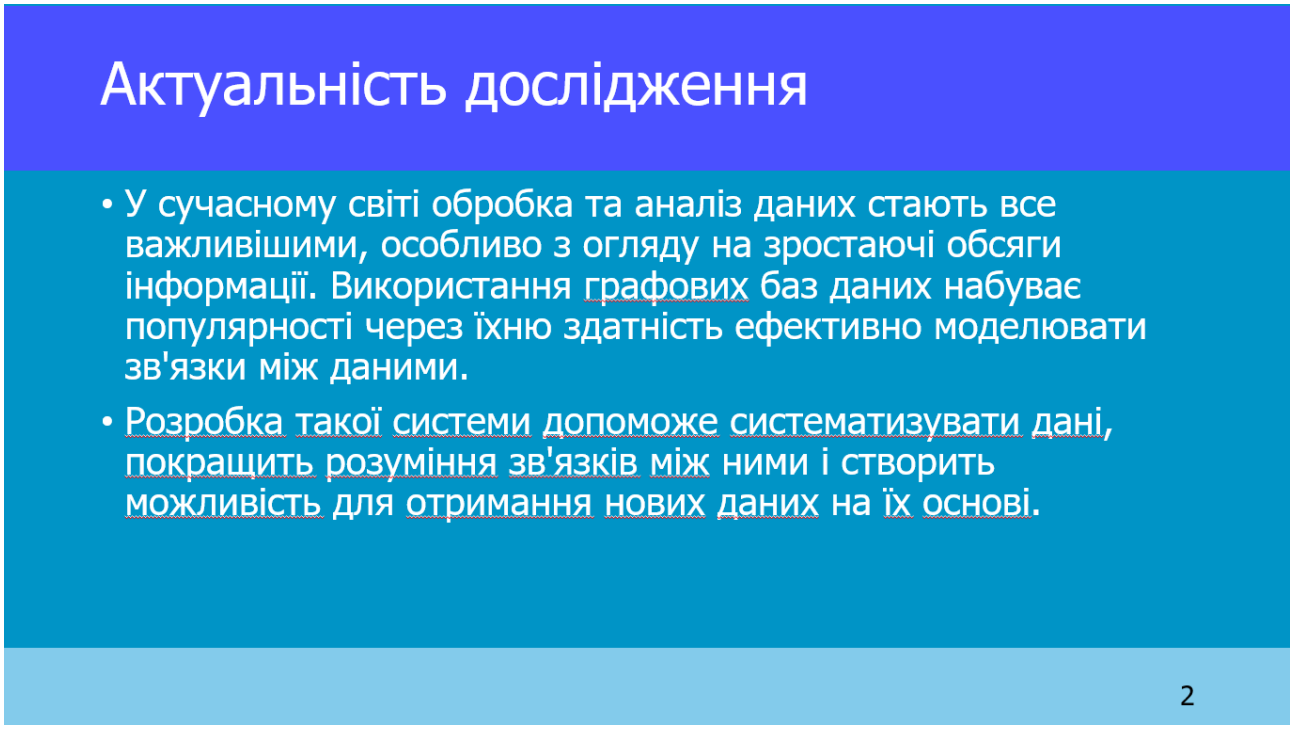


Рисунок Г.2 – Слайд презентації №2

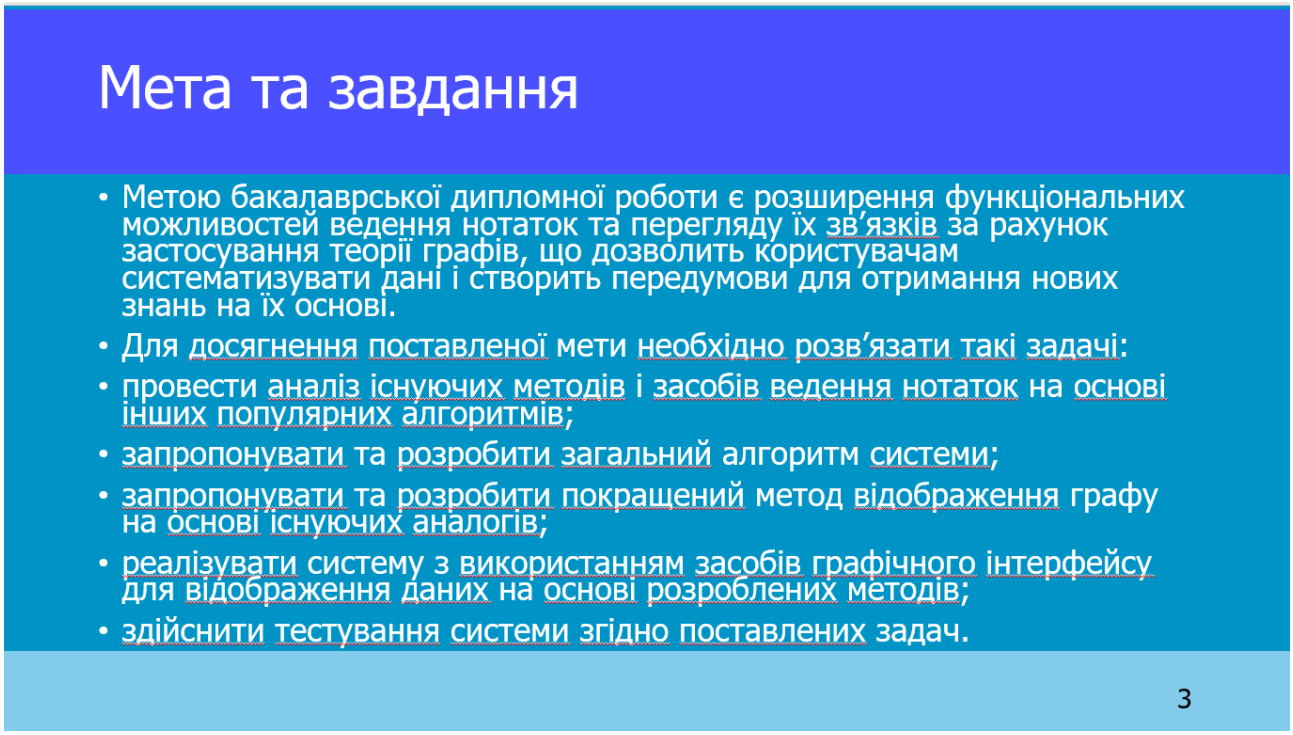


Рисунок Г.3 – Слайд презентації №3

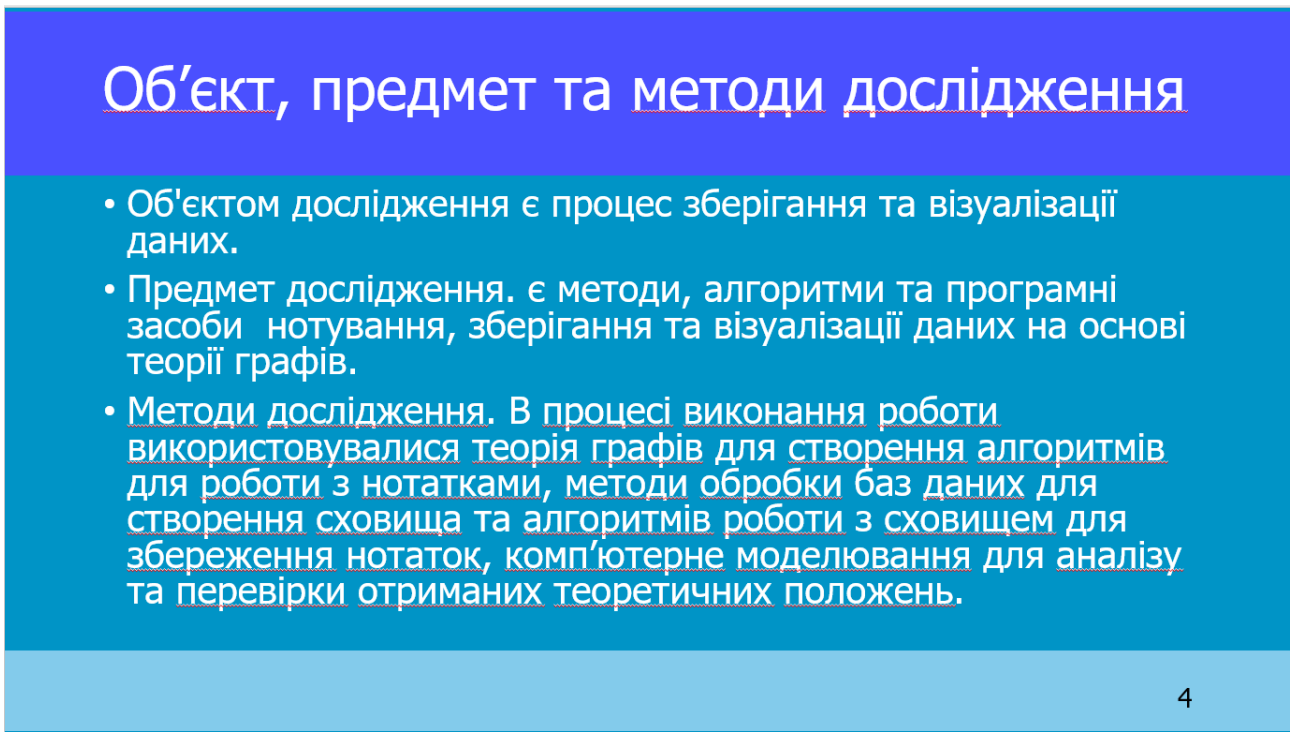


Рисунок Г.4 – Слайд презентації №4

Наукова новизна

- 1. Вдосконалено метод обходу графу в глибину (DFS), який відрізняється від існуючого можливістю обходу ізольованих вершин (гілок) за допомогою лінійного пошуку, що дозволило розширити сферу застосування DFS на графи, що містять кілька деревовидних структур або ізольовані вершини.
- 2. Вдосконалено метод візуалізації графу , в якому на відміну від існуючих, передбачено візуалізацію на основі послідовного перебору всіх вершин з присвоєнням випадкових координат кожній вершині та подальшого створення ребер з використанням зв'язків між вершинами, що дозволило отримати більш точну та зрозумілу візуалізацію структури графу та полегшує загальне розуміння взаємозв'язків між вершинами.

5

Рисунок Г.5 – Слайд презентації №5

Практична цінність отриманих результатів

- Розроблено алгоритми та програми, за допомогою яких користувачі зможуть створювати нотатки та досліджувати їх. Алгоритми та окремі програмні модулі можуть бути використані розробниками для створення інших програмних систем для роботи з графами.

6

Рисунок Г.6 – Слайд презентації №6

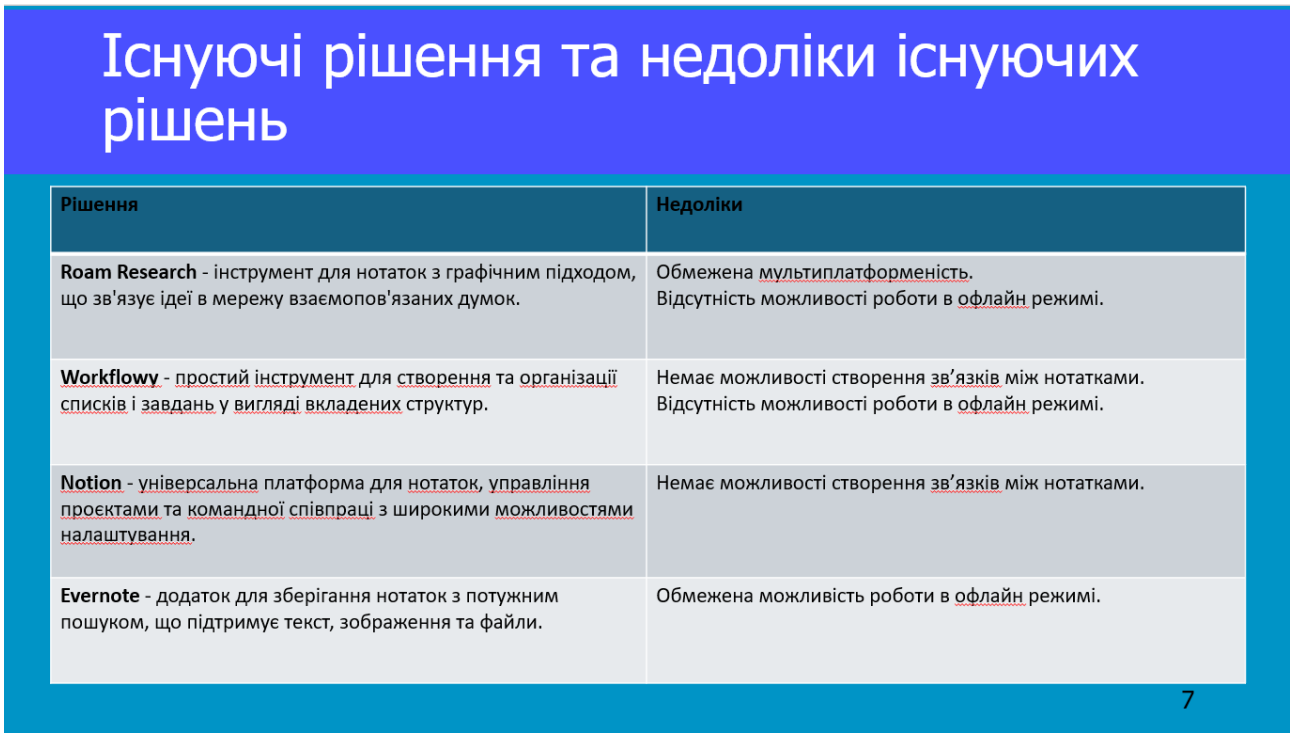


Рисунок Г.7 – Слайд презентації №7

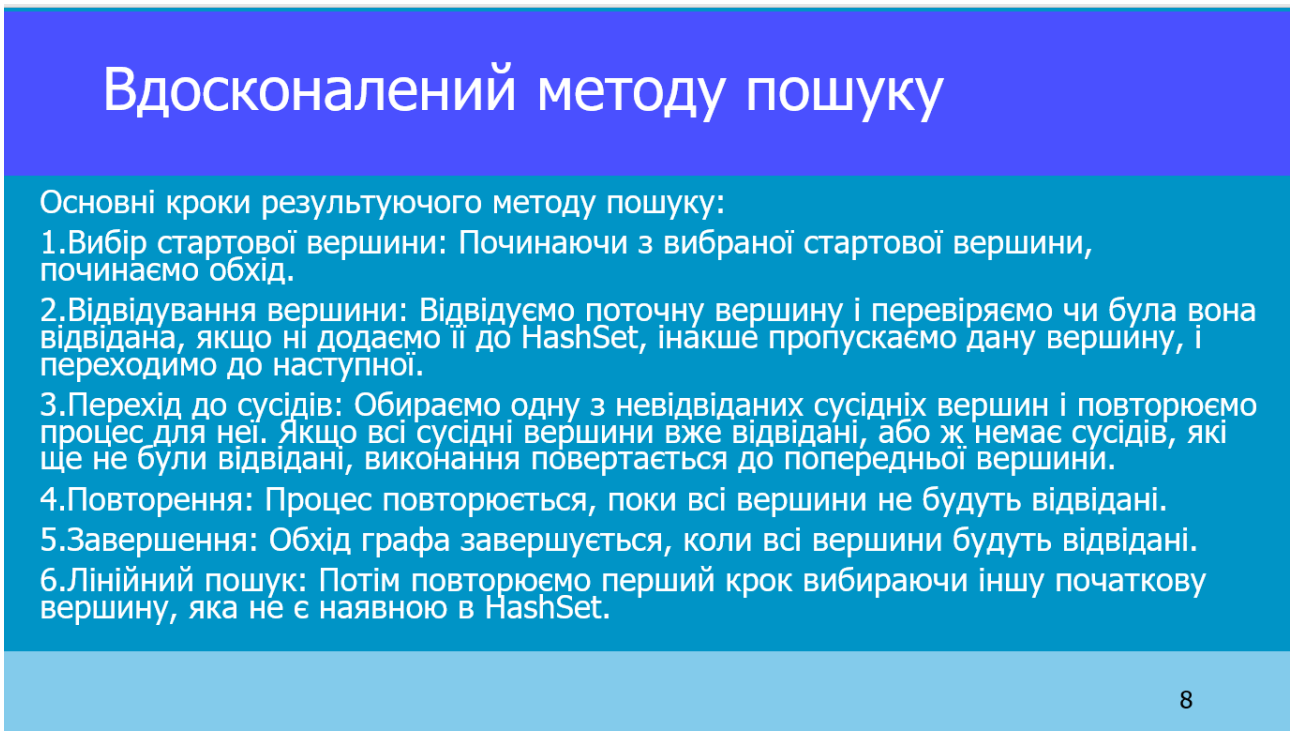
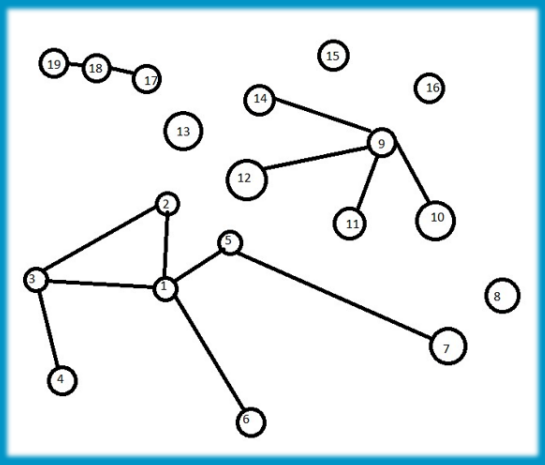


Рисунок Г.8 – Слайд презентації №8

Вдосконалений методу візуалізації

- Результуючий метод відображення графу має нижче наведену послідовність дій:
- 1.Вибір вершини: Вибираємо вершину з списку вершин.
- 2.Присвоєння координат: Присвоєння випадкових координат для кожної вершини, між мінімальними та максимальними, розмірами вікна відображення.
- 3.Перехід до наступної вершини: Переходимо до наступної вершини.
- 4.Візуалізація вершин: Візуалізуємо кожну вершину у вигляді кола за допомогою відповідних інструментів, також підчас візуалізації додаємо кожен зв'язок до HashSet.
- 5.Візуалізація ліній: Візуалізуємо всі лінії за допомогою відповідних інструментів графічної бібліотеки, які присутні в даному HashSet.

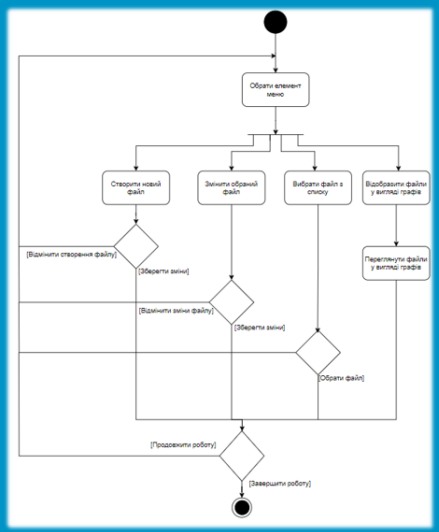


9

Рисунок Г.9 – Слайд презентації №9

Діаграма діяльності додатку

Починаючи з відкриття додатку або входу до системи, діаграма показує послідовність кроків, які користувач повинен виконати для досягнення своїх цілей або виконання конкретної функції. Ці кроки можуть включати введення даних, вибір опцій, натискання кнопок та перегляд реакції системи на кожну дію користувача.



10

Рисунок Г.10 – Слайд презентації №10

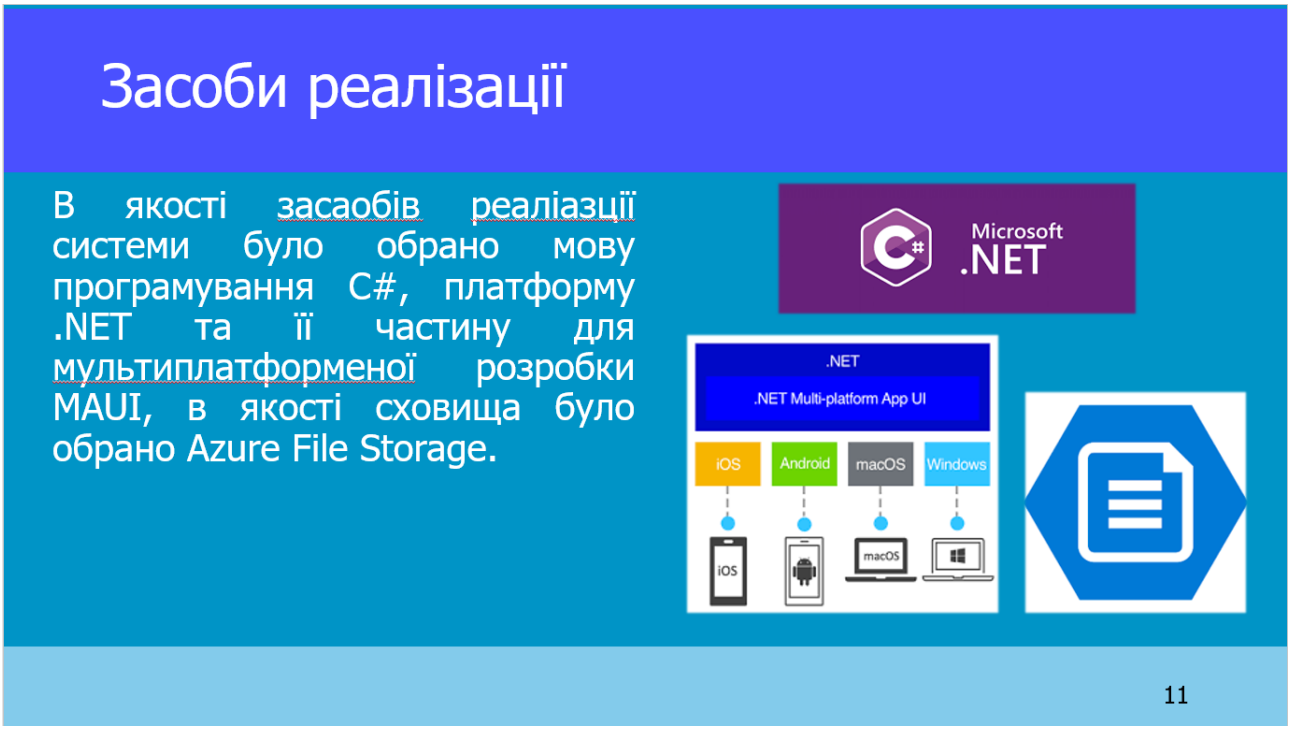


Рисунок Г.11 – Слайд презентації №11

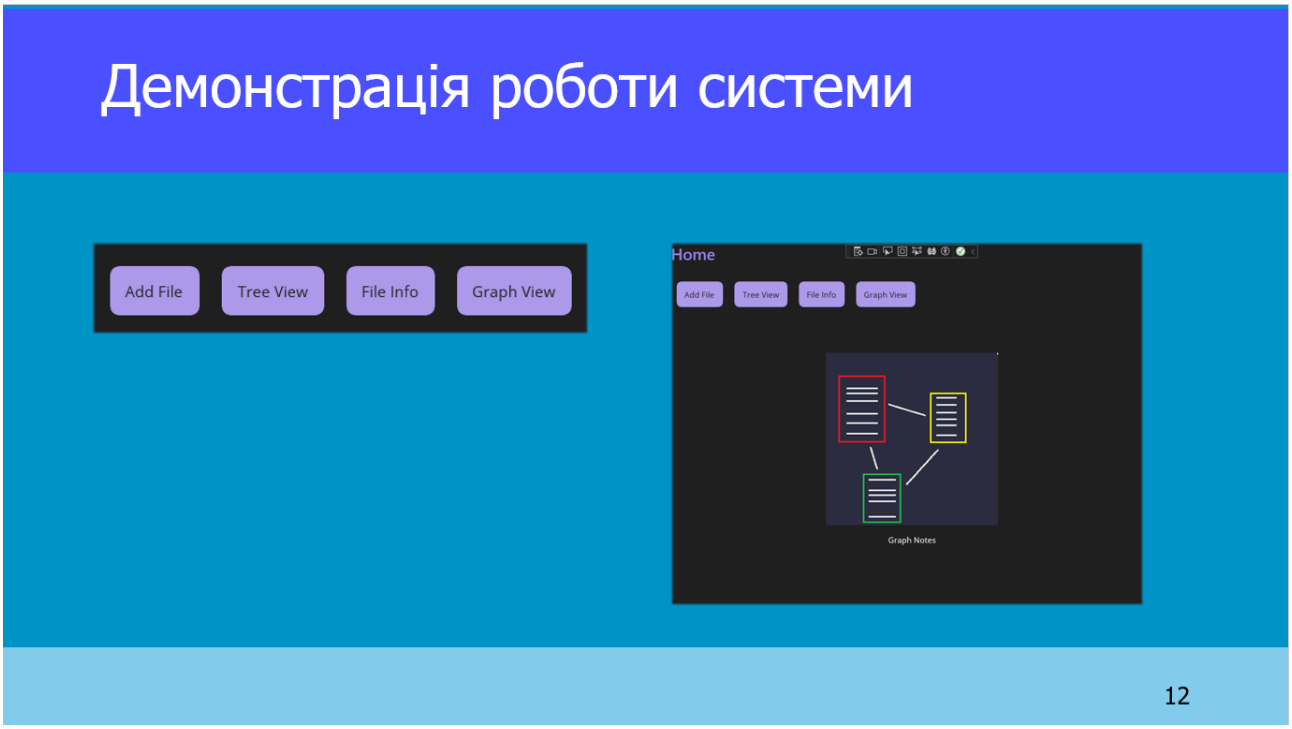


Рисунок Г.12 – Слайд презентації №12

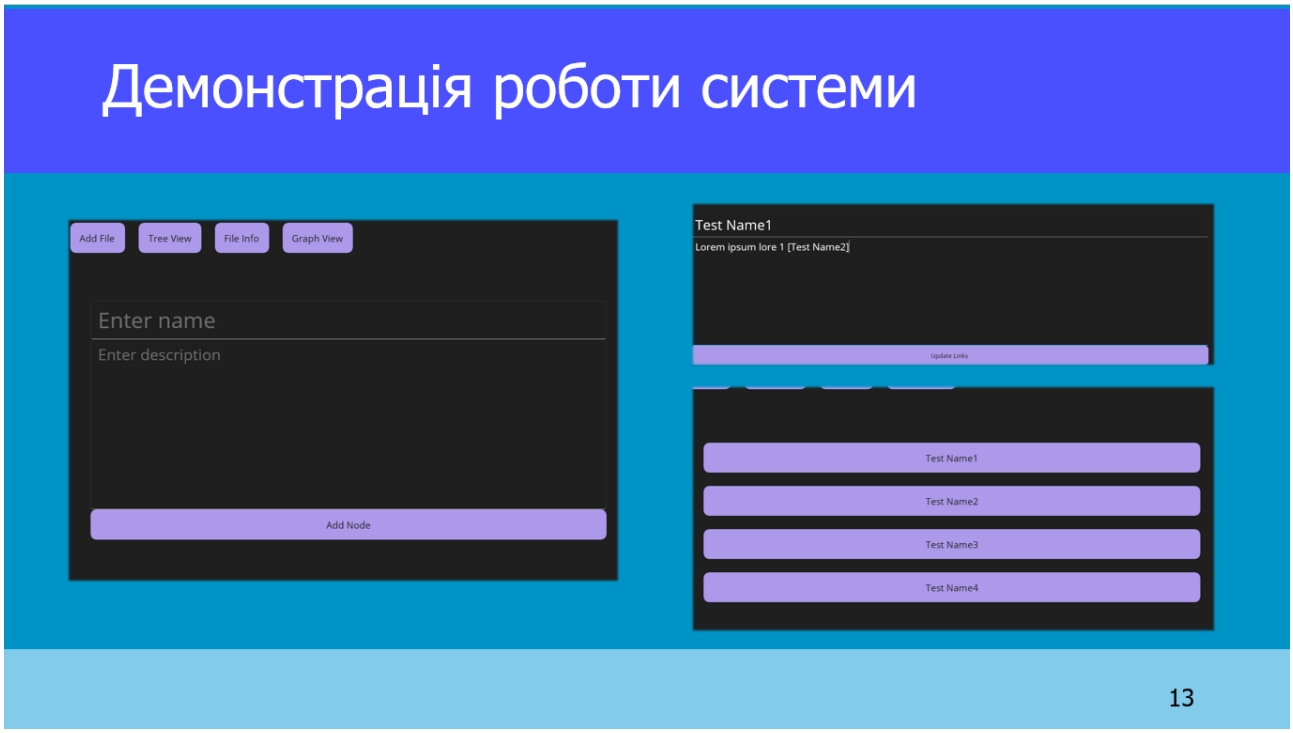


Рисунок Г.13 – Слайд презентації №13

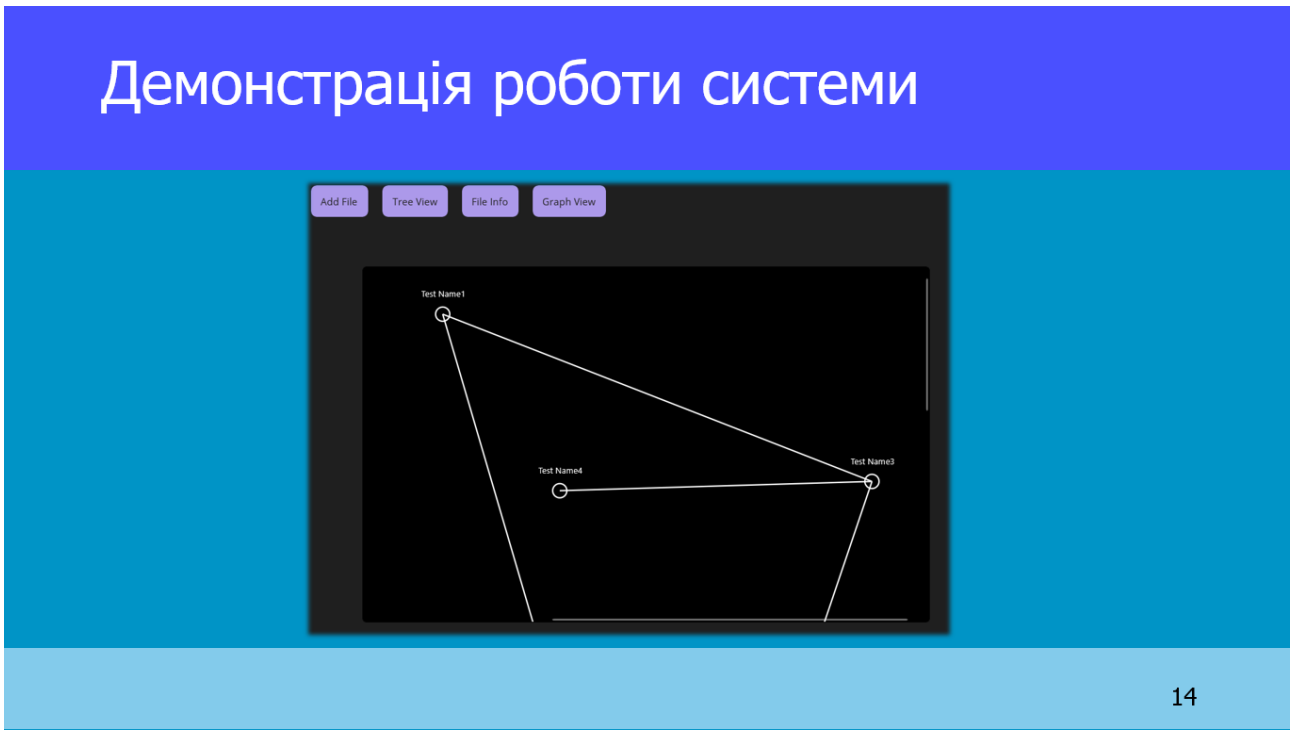


Рисунок Г.14 – Слайд презентації №14

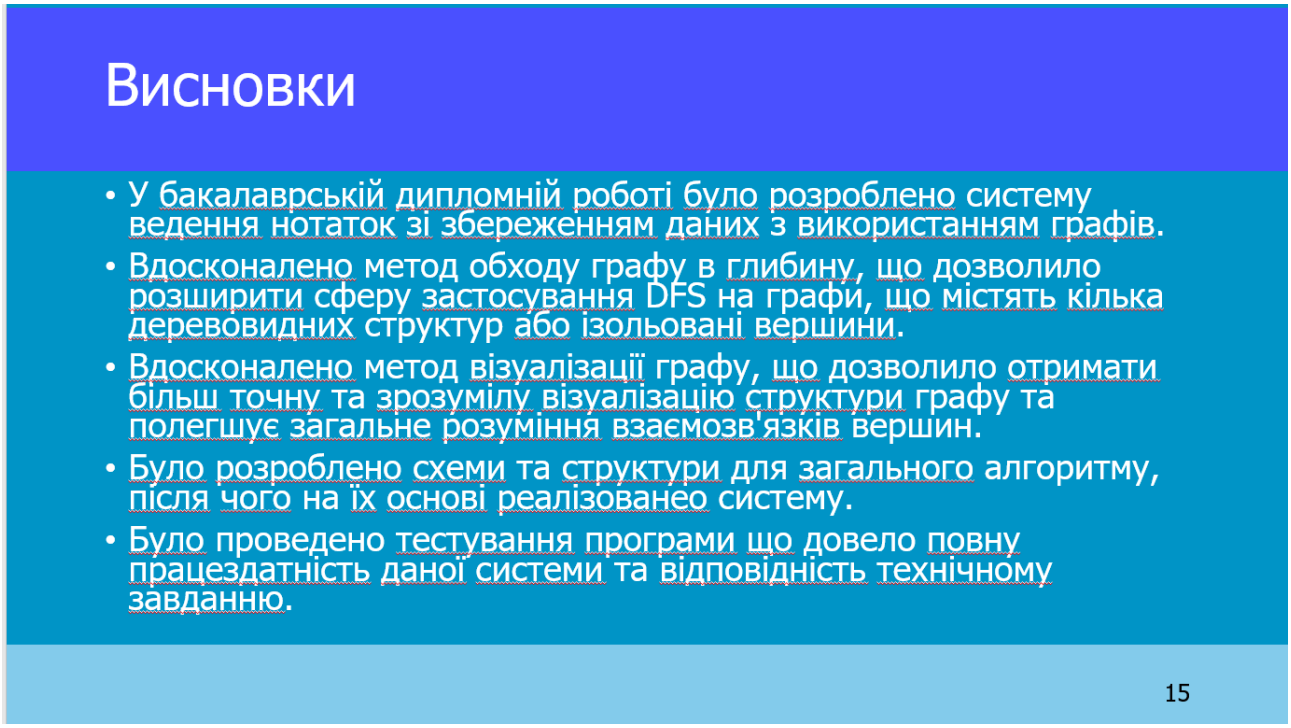


Рисунок Г.15 – Слайд презентації №15

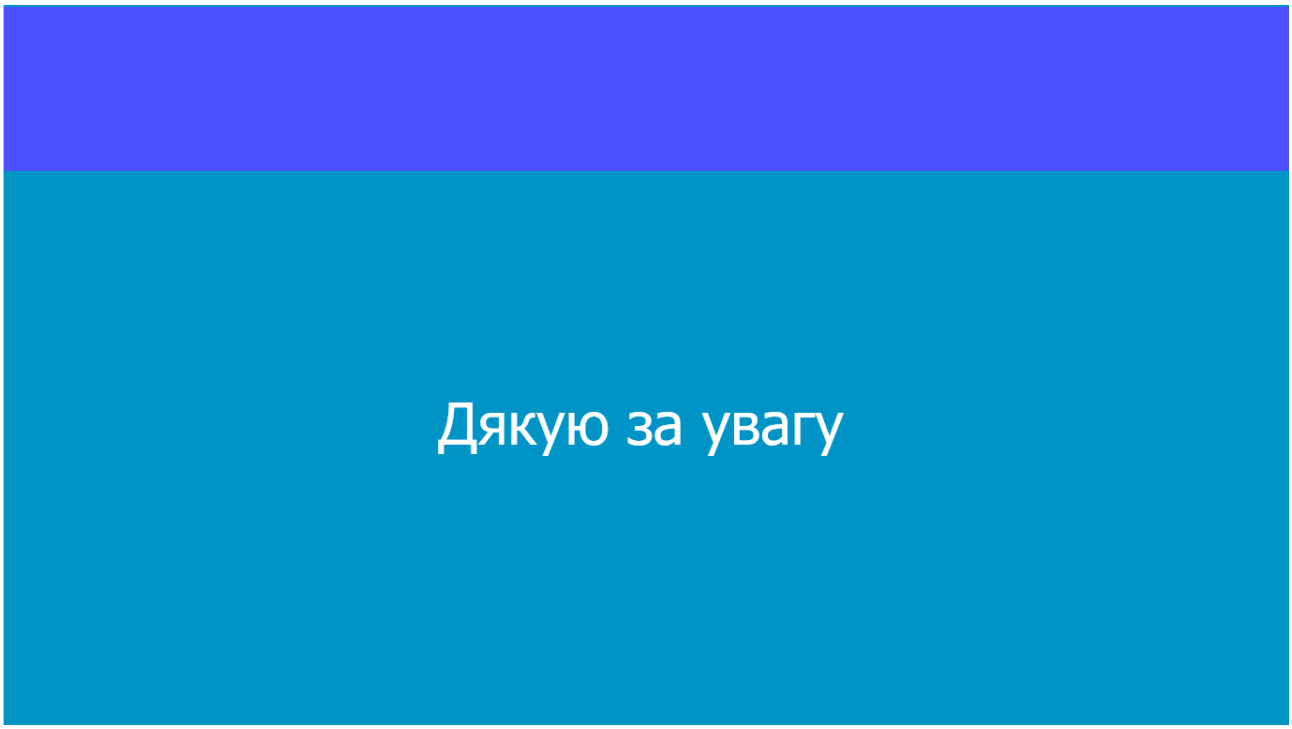


Рисунок Г.16 – Слайд презентації №16