Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів»

Виконав: студент II курсу
групи 2ПІ-23м
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Музичук Д.Р.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Войтко В.В.
(прізвище та ініціали)
«06» грудня 2024 р.
Опонент: к.т.н., доц. каф. ОТ Колесник І.С.
(прізвище та ініціали)
«06» грудня 2024 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«06» грудня 2024 р.

ВНТУ – 2024

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«18» вересня 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Музичуку Дмитру Романовичу

1. Тема роботи – Розробка методу та засобів системи автоматизованого формування і редагування текстових документів.

Керівник роботи: Войтко Вікторія Володимирівна, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від «17» вересня 2024 р. № 310.

2. Строк подання студентом роботи 3 грудня 2024р.

3. Вихідні дані до роботи: середовище розробки Android Studio, мова розробки Kotlin, метод автоматичної генерації структури документа, метод формування рекомендацій щодо наповнення документа, метод рекомендацій елементів візуалізації, метод роботи з елементами візуалізації у Android View інтерфейсі.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану систем для редагування й формування текстових документів, розробка методів; моделі та алгоритмів роботи системи; розробка програмних засобів системи; тестування системи; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми;

мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна практична цінність одержаних результатів; порівняльний аналіз аналогів використані технології, розробка методів; модель системи; загальний алгоритм роботи системи; структура класів даних; тестування системи; висновки апробація результатів роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Войтко В.В., к.т.н., доцент кафедри ПЗ	19.09.2024	02.12.202
5	Причепа І.В., к.е.н., доцент кафедри ЕПВМ	22.11.2024	30.11.202

7. Дата видачі завдання 19 вересня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану систем для редагування й формування текстових документів	20.09.2024 30.09.2024	вик.
2	Розробка методів, моделі та алгоритмів роботи системи	01.10.2024 17.10.2024	вик.
3	Розробка програмних засобів системи	18.10.2024 04.11.2024	вик.
4	Тестування системи	05.11.2024 21.11.2024	вик.
5	Економічна частина	22.11.2024 30.11.2024	вик.

Студент _____ (підпис) Музичук Д.Р. (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ (підпис) Войтко В.В. (прізвище та ініціали)

Анотація

УДК 004.912.032.26

Музичук Д.Р. Розробка методу та засобів системи автоматизованого формування і редагування текстових документів. Магістерська кваліфікаційна робота зі спеціальності 121 – Інженерія програмного забезпечення, освітня програма – Інженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 108 с.

На укр. мові. Бібліогр.: назв: 33; рисунків 47; таблиць: 8.

У магістерській кваліфікаційній роботі подано результати дослідження інструментів для автоматизованого редагування й формування текстових документів. Обґрунтовано доцільність удосконалення методів роботи зі структурою документа, його вмістом та елементами візуалізації.

Магістерська кваліфікаційна робота містить аналіз відомих підходів до автоматизованого редагування й формування текстових документів, методів формування структури документа та його вмісту.

Розроблено методи формування структури документа, рекомендацій наповнення документа, рекомендацій елементів візуалізації, роботи з елементами візуалізації у Android View інтерфейсі текстового редактора.

Розроблено систему для автоматизованого редагування й формування текстових документів, яка включає функціонал для автоматичної генерації структури документа, формування рекомендацій щодо наповнення документа вмістом, формування рекомендацій щодо елементів візуалізації на основі введеного користувачем тексту, додавання таблиць, зображень та графіків до документа. Документи зберігаються в локальному сховищі пристрою користувача за допомогою бібліотеки Room, яка відповідає за забезпечення бази даних для застосунку та збереження даних.

Ключові слова: редагування текстових документів, автоматизація, Android View.

Abstract

Muzychuk D.R. Development of a method and means of a system for automated formation and editing of text documents. Master's qualification work in the specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2024. 108 pages.

In Ukrainian. Bibliography: titles: 33; figures 47; tables: 8.

The master's qualification work presents the results of research into tools for automated editing and formation of text documents. The feasibility of improving methods for working with the structure of a document, its content and visualization elements is substantiated.

The master's qualification work contains an analysis of known approaches for automated editing and formation of text documents, methods for forming the structure of a document and its content.

Methods for forming the structure of a document, recommendations for filling the document, recommendations for visualization elements, and working with visualization elements in the Android View interface of a text editor are developed.

A system for automated editing and generation of text documents has been developed, which includes functionality for automatic generation of document structure, generation of recommendations for filling the document with content, generation of recommendations for visualization elements based on the text entered by the user, addition of tables, images and graphs to the document. Documents are stored in the local storage of the user's device using the Room library, which is responsible for providing the database for the application and saving data.

Keywords: editing text documents, automation, Android View.

ЗМІСТ

ВСТУП

Актуальність роботи. Робота з документами є невід'ємною частиною будь-якого робочого процесу в різних галузях, починаючи від бізнесу та закінчуючи науковою діяльністю.

Поява документів бере свій початок із появи поняття ділових відносин близько 3000 років до н.е. [1].

Винайдення паперу сприяло значному зростанню обсягу документів завдяки своїй зручності, а також зростанню потреби у створенні документів, що пов’язано із розвитком цивілізації.

Промислова революція принесла друковані документи, зростання їхньої кількості. Була винайдена файлова система для їх упорядкування, до якої входили картотеки та архіви.

У XX столітті, з появою електронних обчислювальних машин, а потім і персональних комп’ютерів, ведення документації стрімкими темпами перейшло у цифровий простір, адже це дозволяло зручніше працювати із документами, їх зберігати та їх обробляти.

Значний поштовх у розвиток документообігу задає сучасна інформаційна епоха, яка генерує велику кількість знань та інформації, яку необхідно відповідним чином оформити та впорядкувати для ефективного її використання.

Документи використовуються для збереження, передачі та представлення інформації, тому важливо, щоб процес їх створення був максимально ефективним, точним і зручним для користувача. З огляду на швидкий розвиток технологій і необхідність обробляти великі обсяги інформації, автоматизація роботи з документами стає дедалі актуальнішою. Впровадження інструментів для автоматичного створення та редагування документів дозволяє зекономити час, уникнути помилок і зробити документи більш професійними та структурованими. Тому актуальною є розробка системи автоматизованого формування і редагування текстових документів.

Зв’язок роботи з науковими програмами, планами, темами. Робота

виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою магістерської кваліфікаційної роботи є підвищення рівня автоматизації процесу розробки текстових документів шляхом розробки та впровадження системи автоматизованого формування і редагування документів, що дозволить підвищити якість та надійність обробки текстових файлів.

Основними завданнями дослідження є:

– розробити модель, методи та алгоритми роботи системи для автоматичного формування і редагування текстових документів;

– розробити функціонал для автоматизованого структурування документів;

– розробити функціонал для інтерактивних порад щодо наповнення документів вмістом;

– розробити функціонал для порад щодо наповнення документів релевантним додатковим вмістом (наприклад таблиці, зображення, графіки);

– налаштувати застосунок для роботи на різних видах пристроїв;

– провести тестування розробленої системи.

Об’єктом дослідження є процес розробки системи для редагування та формування текстових документів.

Предметом дослідження є методи та засоби розробки системи для редагування та формування текстових документів.

Методи дослідження. У процесі досліджень використовувались методи дослідження:

– методи розробки застосунку під операційну систему Android для реалізації системи для редагування й формування текстових документів;

– методи розробки інтерфейсу системи для редагування й формування текстових документів;

– методи штучного інтелекту для інтеграції засобів штучного інтелекту в систему;

– методи роботи з елементами візуалізації у Android для реалізації інтерфейсу системи.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод формування структури документа, який, на відміну від відомих, використовує штучний інтелект для формування розділів та структурних частин документа, враховуючи його тему та призначення, що дозволить автоматизувати процес редагування документів та підвищити надійність процесу формування текстової документації.

2. Подальшого розвитку отримав метод рекомендацій наповнення документа, який, на відміну від відомих, використовує штучний інтелект для надання точних рекомендацій залежно від теми та призначення документа, що дозволить підвищити ефективність створення текстових документів.

3. Подальшого розвитку отримав метод рекомендацій елементів візуалізації, який, на відміну від відомих, відстежує зміни документа та додає елементи візуалізації для підвищення інформативності документа та згідно з його контекстом, що дозволить покращити інтерфейсні можливості системи в процесі автоматизованого формування текстових документів.

4. Подальшого розвитку отримав метод роботи з елементами візуалізації у Android View інтерфейсі текстового редактора, який, на відміну від існуючих, використовує діалогові вікна та конвертацію елементів візуалізації у зображення, що дозволить покращити інтерфейсні можливості системи у процесі взаємодії з користувачем.

Практична цінність отриманих результатів. Розроблена система може використовуватися для прискорення та полегшення процесу формування й редагування текстових документів.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто. У науковій роботі [2], опублікованій у співавторстві, автору належать такі результати: загальний алгоритм роботи системи, таблиця порівняння аналогів.

Апробація матеріалів. Описані у магістерській кваліфікаційній роботі

положення доповідалися на міжнародній науково-практичній конференції «Інформаційні технології та автоматизація – 2024».

Публікації. Результати роботи опубліковані в тезах доповіді [2] на міжнародній науково-практичній конференції «Інформаційні технології та автоматизація – 2024».

Структура та обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, п’яти розділів, висновків, списку використаних джерел, що містить 34 найменування, 4 додатків. Робота містить 47 ілюстрації, 8 таблиць.

1 АНАЛІЗ СТАНУ СИСТЕМ ДЛЯ РЕДАГУВАННЯ Й ФОРМУВАННЯ ТЕКСТОВИХ ДОКУМЕНТІВ

1.1 Аналіз стану систем роботи з текстовими документами

Сучасні системи для редагування й формування текстових документів надають широкий функціонал роботи з різного роду документацією. Звичним уже є робота з діаграмами всередині текстового редактора, голосовий ввід, макроси, які створюють специфічні шаблони документів, відстежуючи дії користувача.

З постійним зростанням обсягів знань і даних, зростає і потреба у їх документуванні для їх розуміння та впорядкування. Цей процес займає все більше часу та вимагає все більше ресурсів. При цьому, різного роду документація має власні вимоги до оформлення, що є важливим у різних сферах. При цьому, стандартизація оформлення відкриває можливості до автоматизації процесу форматування тексту задля того, аби приділити більше уваги самому змісту. Тому, все більшу необхідність отримують системи, що автоматизують процес редагування й формування текстових документів [3].

На сьогодні існує, наприклад, програмне забезпечення для створення документів, яке дозволяє автоматично створювати звіти або контракти на основі вхідних даних або шаблонів.

Інструменти автоматизації шаблонів також мають велике значення. Вони використовують попередньо визначені шаблони для створення узгоджених і стандартизованих документів, що важливо в таких галузях, як юридична та сфера охорони здоров’я, де уніфікованість і відповідність мають вирішальне значення. Інструменти автоматичного форматування застосовують узгоджені стилі форматування до документів, забезпечуючи дотримання вимог до оформлення.

У бізнес-секторі автоматизовані інструменти для створення та редагування документів спрощують робочі процеси, зменшують кількість помилок, що виникають вручну, і підвищують продуктивність. Компанії можуть

швидко створювати рахунки-фактури, контракти та звіти, що дозволяє співробітникам зосередитися на стратегічних завданнях. В сфері освіти викладачі та студенти отримують переваги від інструментів, які допомагають створювати добре структуровані документи, форматувати цитати та перевіряти вміст на плагіат, тим самим покращуючи якість академічної роботи. Юридична галузь покладається на автоматизацію, щоб гарантувати, що юридичні документи відповідають строгим вимогам щодо форматування та вмісту, зменшуючи ризик невідповідності та прискорюючи підготовку матеріалів справи. У сфері охорони здоров’я медпрацівники використовують автоматизовані інструменти для транскрипції нотаток пацієнтів, створення звітів і забезпечення відповідності документації нормативним стандартам.

Таким чином, автоматизоване формування та редагування текстових документів у сучасному світі перетворилося зі зручності на необхідність. Розробка додатків у цій сфері відповідає нагальним потребам різних секторів щодо ефективного управління інформацією. Оскільки технологія продовжує розвиватися, ці інструменти стануть ще більш інтегрованими, стимулюючи ефективність, сприяючи співпраці та дозволяючи організаціям швидко адаптуватися в динамічному середовищі.

Отже, актуальною є розробка системи автоматизованого формування і редагування текстових документів.

1.2 Аналіз аналогів розроблюваного програмного застосунку

Розглянемо програми, що допомагають в роботі з текстовими документами:

– Microsoft Word;

– Formstack Documents;

– Docmosis;

– Jasper.

Microsoft Word [4] – це популярна програма для створення, редагування та форматування текстових документів. Вона є частиною офісного пакета

Microsoft Office, який використовується як у професійному, так і в особистому житті мільйонів користувачів по всьому світу. Microsoft Word підтримує різні операційні системи, такі як Windows і macOS, а також має версію для мобільних пристроїв.

Однією з головних переваг Microsoft Word є його широкий функціонал. Користувачі можуть створювати документи будь-якої складності, використовуючи різноманітні інструменти для форматування тексту, вставки зображень, графіків, таблиць і навіть відео. Програма підтримує велику кількість шрифтів, стилів абзаців і тем, що дозволяє робити документи не лише інформативними, але й естетично привабливими, а також використовувати функції перевірки правопису та граматики, що значно полегшує процес редагування тексту.

Інтерфейс Microsoft Word наведено на рисунку 1.1.

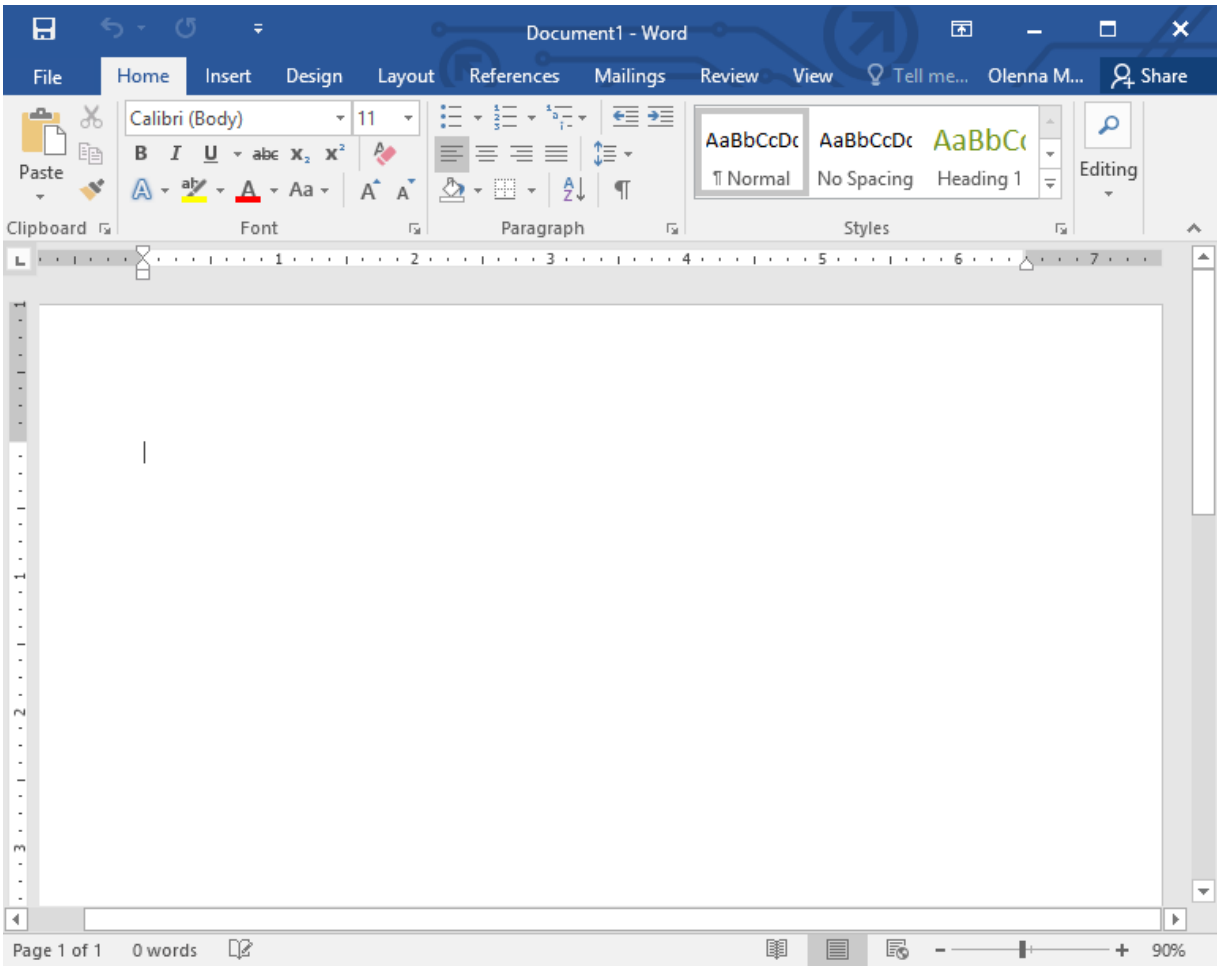

Рисунок 1.1 – Інтерфейс Microsoft Word

Також, користувачі можуть редагувати один і той самий документ одночасно, залишати коментарі та відстежувати зміни, що робить програму незамінною для командної роботи. Word інтегрований із хмарним сервісом OneDrive, що дозволяє зберігати документи онлайн і отримувати доступ до них з будь-якого пристрою. Це забезпечує зручність і мобільність для користувачів, які часто працюють у відрядженнях або на віддаленій основі.

Microsoft Word також надає широкий вибір шаблонів для створення різноманітних документів: резюме, звітів, листів, брошур, презентацій тощо. Це значно економить час, так як шаблони допомагають структурувати документи, дотримуючись професійного вигляду, що особливо корисно для офіційних документів та бізнес-комунікацій.

Formstack Documents [5] – це платформа для автоматизації створення та управління документами, що допомагає підвищити ефективність та зменшити ручну роботу, пов’язану з документами.

Цей програмний засіб дозволяє автоматично генерувати документи на основі попередньо створеного шаблону, що може бути корисно для створення контрактів, рахунків, листів та подібних стандартизованих видів документів. Користувачі можуть створювати самостійно потрібні їм шаблони документів у відповідному редакторі.

Застосунок підтримує різні формати документів, які є найбільш поширеними у документообігу. Платформа забезпечує безпечне надсилання документів через електронну пошту чи хмарні сховища за допомогою електронного підпису. Formstack Documents дозволяє налаштовувати складні робочі процеси з використанням умовної логіки, що забезпечує більш гнучкий та персоналізований підхід до створення документів. Автоматизація рутинних монотонних завдань дозволяє зосередити час та зусилля на більш важливі аспекти ведення ділової діяльності, що значно підвищує продуктивність праці. Також, це дозволяє знизити ризик виникнення помилок через людський фактор, швидко та точно оформляти та надавати необхідні документи.

Інтерфейс Formstack Documents наведено на рисунку 1.2.

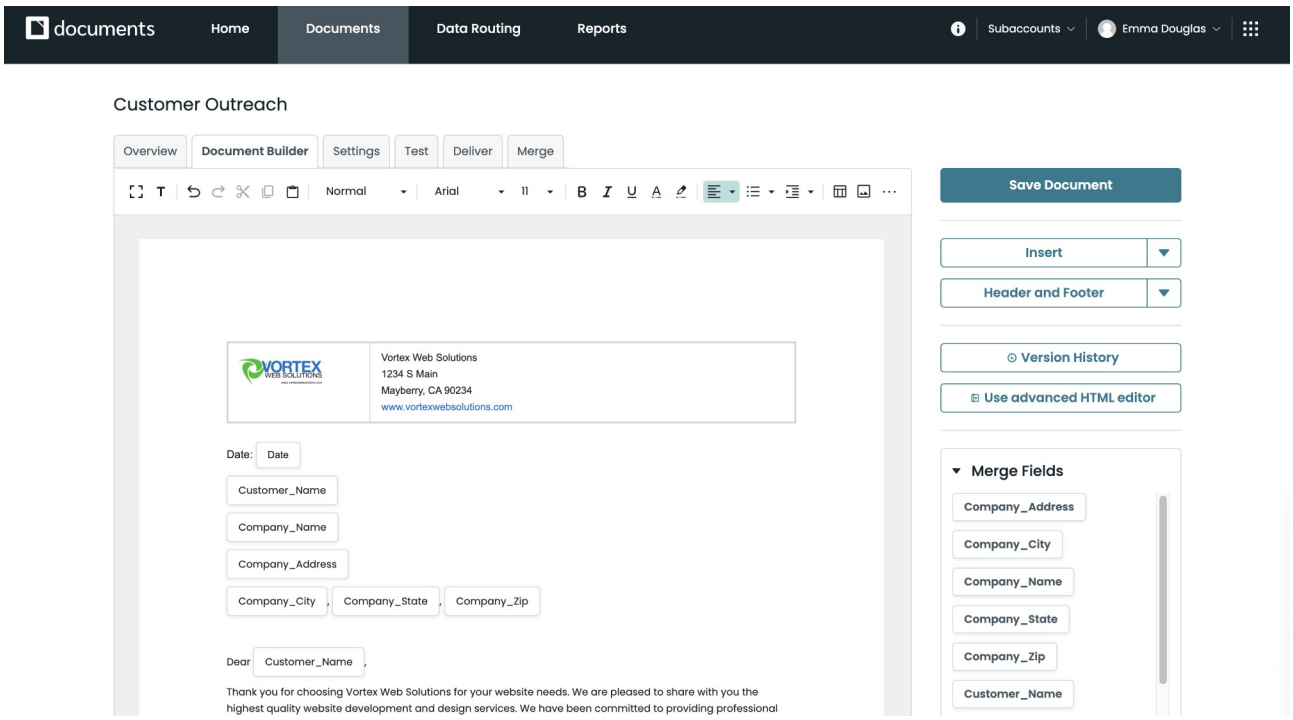

Рисунок 1.2 – Інтерфейс Formstack Documents

Docmosis [6] – це платформа для генерації документів та звітів, яка дозволяє автоматизувати процес створення документів, використовуючи шаблони Microsoft Word або LibreOffice, що дозволяє використовувати користувачам уже відомі програмні засоби для налаштування шаблонів.

Docmosis інтегрується з різними мовами програмування та фреймворками з метою легкого додавання функціоналу генерації документів у свої застосунки. Шаблони можуть бути наповнені даними з різних джерел, включаючи API, бази даних, файли XML або JSON. Пропонується хмарний сервіс Docmosis Cloud для збереження та обігу документів. Також, як альтернативний спосіб, реалізовано функціонал для розгортання необхідного функціоналу на власних серверах локально.

Docmosis дозволяє зменшити витрати на створення шаблонів, скоротити час, що витрачається на це, та ресурси, що потрібно витратити на процеси документообігу. Платформа підтримує шифрування даних при передачі та зберіганні, а також відповідає міжнародним стандартам безпеки. Це забезпечує захист конфіденційної інформації та відповідність нормативним вимогам.

Опція локального розгортання дозволяє компаніям зберігати всі дані

всередині власної інфраструктури, що є критично важливим для багатьох галузей з високими вимогами до конфіденційності.

Інтерфейс Docmosis наведено на рисунку 1.3.

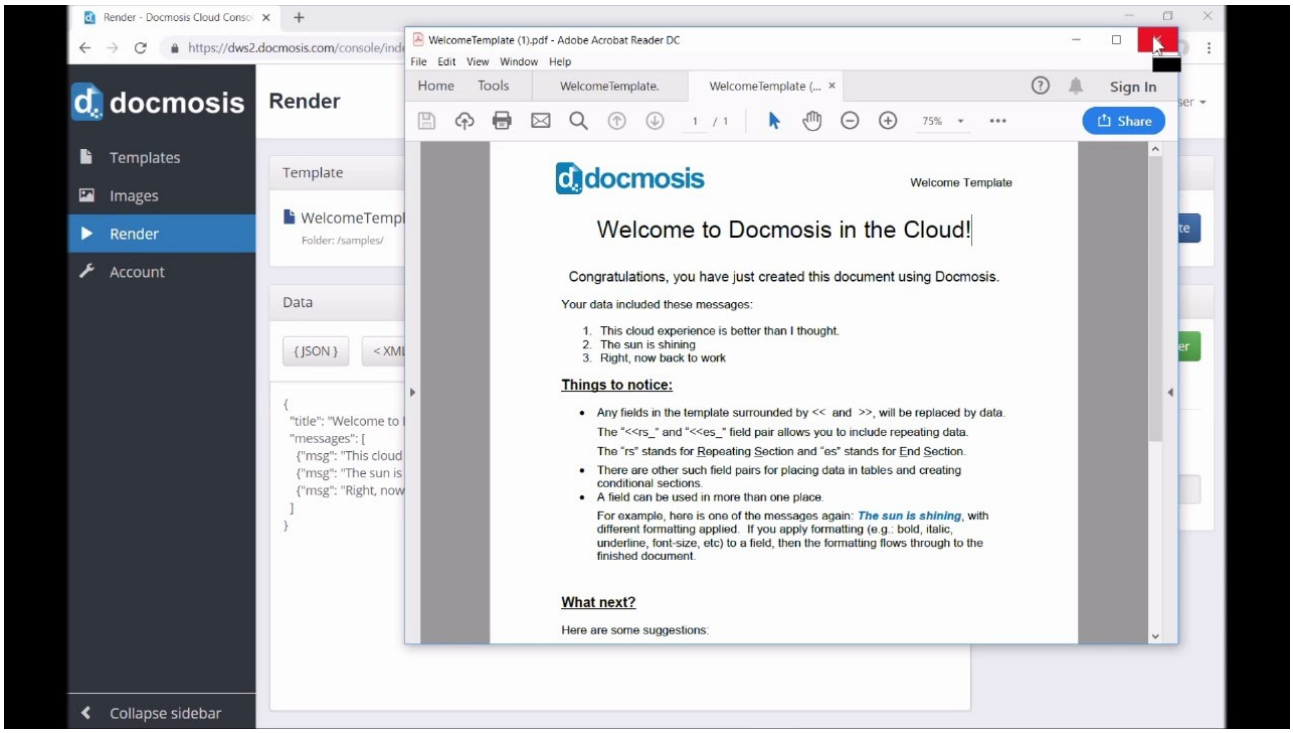

Рисунок 1.3 – Інтерфейс Docmosis

Jasper [7] – це платформа на основі штучного інтелекту, розроблена для створення контенту, включаючи текстові матеріали, маркетингові тексти, блоги, соціальні пости, сценарії для відео та багато іншого. Завдяки використанню технології штучного інтелекту, Jasper здатний генерувати високоякісний контент за короткий час, що особливо корисно для маркетологів, копірайтерів та інших фахівців, яким потрібно регулярно створювати тексти. Платформа дозволяє автоматизувати рутинні завдання, пов'язані з написанням контенту, значно зменшуючи час, який потрібно на його створення.

Однією з ключових функцій Jasper є можливість генерувати тексти на основі мінімальних вхідних даних, яка полягає у тому, що користувач вводить короткий опис або ключові ідеї, а система автоматично створює відповідний текст, враховуючи стиль написання, тон і мету тексту, роблячи його

адаптованим до певної аудиторії або маркетингової стратегії.

Інтерфейс Jasper наведено на рисунку 3.4.

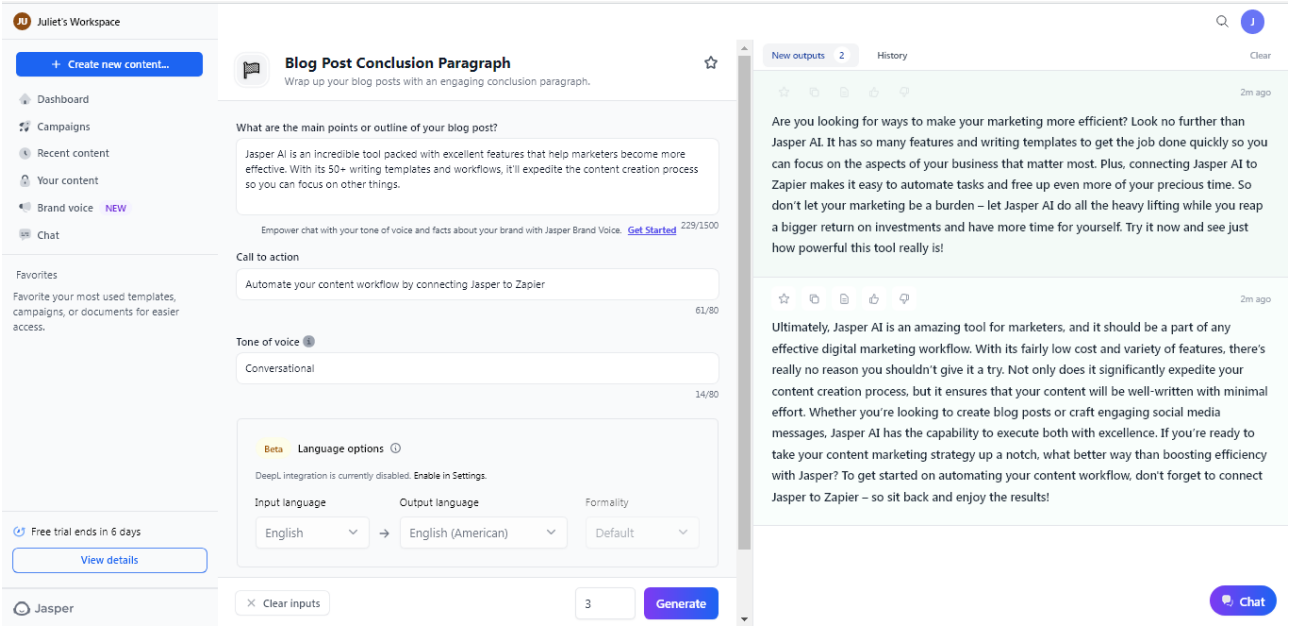

Рисунок 1.4 – Інтерфейс Jasper

Jasper пропонує широкий спектр інструментів для вдосконалення контенту. Окрім базового текстового генератора, він має функції для перевірки граматики, стилістики, а також аналітики читабельності, що дозволяє користувачам налаштовувати текст, коригувати його структуру та отримувати рекомендації щодо покращення. Особливий акцент зроблено на SEO-оптимізацію: Jasper може генерувати тексти, враховуючи ключові слова та інші фактори, що впливають на пошукову видимість, роблячи його корисним інструментом для цифрового маркетингу.

Отже, Jasper – це потужний інструмент для автоматизації створення текстового контенту, який пропонує значні переваги в продуктивності та якості текстів. Його можливості варіюються від базової генерації тексту до просунутої SEO-оптимізації та налаштування стилю, що робить платформу корисною для бізнесів різного масштабу. Завдяки AI-технологіям, Jasper дозволяє швидко створювати якісний контент, адаптований до конкретних цілей і аудиторій, що значно полегшує роботу копірайтерів і маркетологів.

Щоб наочно продемонструвати відмінності між цими додатками, була створена порівняльна таблиця (таблиця 1.1).

Таблиця 1.1 – Порівняння характеристик аналогів

	Microsoft Word	Formstack Documents	Docmosis	Jasper	Власна розробка
Перевірка граматики	1	1	0	1	1
Текстовий редактор	1	0	1	1	1
Шаблони для документів	1	0	1	0	1
Динамічні рекомендації елементів візуалізації у документі	1	0	0	0	1
Поради щодо текстового наповнення документа	1	1	0	1	1
Динамічне структурування документа	0	0	1	0	1
Загальна оцінка	4	2	3	3	6

Виходячи з цієї таблиці порівняння, можна зробити висновок, що запропонована власна розробка є актуальною, має переваги перед існуючими аналогами.

1.3 Аналіз методів розв’язання задачі

Один із основних методів розв’язання задачі полягає у розробці текстового редактора, який надаватиме функціонал користувачеві для редагування тексту й роботи з елементами візуалізації, з використанням штучного інтелекту як асистента при написанні документа. Такий метод дозволяє частково автоматизувати та спростити процес формування й редагування текстового документа, при цьому, залишає контроль користувача над розробкою, запобігає академічному плагіату, та застосовує штучний

інтелект як допоміжний засіб при розробці документації.

Цей метод передбачає надання основних відомостей користувачем про документ, що розробляється, а штучний інтелект створює основну структуру документа та поради, як краще його реалізувати.

Інший метод полягає у використанні алгоритмів класифікації тексту, який навчається на наборах даних, що містять приклади різних типів документів, таких як звіти, листи, рахунки-фактури або юридичні контракти. Вивчаючи особливості, які відрізняють кожен тип, модель може класифікувати нові документи на основі їх вмісту. Коли користувач вводить текст, система аналізує його, щоб передбачити тип документа та автоматично вибирає шаблон, який відповідає визначеній категорії [8].

Ще один метод, що існує, передбачає вилучення метаданих і аналіз вмісту – система перевіряє такі елементи, як назва документа, заголовки, ключові слова та навіть стиль написання, щоб отримати підказки щодо його мети та теми. Наприклад, часте використання технічного жаргону може вказувати на науковий звіт, спонукаючи до вибору відповідного шаблону.

Після вибору відповідного шаблону система може автоматично генерувати рекомендації щодо того, що слід включити в кожен розділ нового документа. Це досягається кількома підходами:

1. Шаблони розроблено з вбудованими підказками або заповнювачами, які адаптуються відповідно до теми документа. Наприклад, у шаблоні проектної пропозиції система може запропонувати включити конкретні цілі або методології, що стосуються визначеної сфери.

2. Розширені мовні моделі аналізують контекст документа, щоб надати вказівки для окремих розділів. Ці моделі можуть генерувати контури або маркери, які виділяють ключові області, які потрібно розглянути в кожній частині документа. Наприклад, якщо документ є маркетинговим планом, система може рекомендувати обговорити цільові демографічні показники, маркетингові канали та розподіл бюджету.

3. Шляхом підключення до доменних баз даних або сховищ знань система

може пропонувати детальні рекомендації. У разі юридичного контракту система може залучати стандартні пункти або правові положення, які зазвичай використовуються для такого типу угод.

4. Система вивчає попередні документи та дані користувачів, щоб уточнити свої рекомендації. Якщо певні розділи часто включаються в подібні документи, система запропонує їх додати, адаптувавши поради на основі минулих уподобань і організаційних стандартів.

5. Деякі системи використовують інтерактивні елементи, які задають користувачеві цільові запитання. Надані відповіді допомагають системі генерувати рекомендації щодо вмісту для кожного розділу. Цей метод гарантує, що вся необхідна інформація буде зібрана та включена в документ.

Ці методи підвищують ефективність створення документів за рахунок зменшення ручних зусиль, необхідних для вибору шаблонів і визначення вмісту для кожного розділу. Вони забезпечують узгодженість документів і допомагають користувачам створювати вичерпний і релевантний вміст, який відповідає меті та аудиторії документа.

1.4 Постановка задач дослідження

Провівши аналіз методів розв’язання поставленої задачі, аналіз стану застосунків для автоматизованого формування і редагування текстових документів, порівняння аналогів було визначено такі завдання дослідження:

– розробити модель, методи та алгоритми роботи системи для автоматичного формування і редагування текстових документів;

– розробити функціонал для автоматизованого структурування документів;

– розробити функціонал для інтерактивних порад щодо наповнення документів вмістом;

– розробити функціонал для порад щодо наповнення документів релевантним додатковим вмістом (наприклад таблиці, зображення, графіки);

– налаштувати застосунок для роботи на різних видах пристроїв;

– провести тестування розробленої системи.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У першому розділі було проведено аналіз стану питання розробки системи для автоматизованого формування і редагування текстових документів, порівняльний аналіз аналогів: Microsoft Word, Formstack Documents, Docmosis, Jasper.

Було проведено аналіз методів розв’язання задачі дослідження. На основі аналізу було проведено постановку задач дослідження, прийнято рішення розробити систему для автоматизованого формування і редагування текстових документів.

2 РОЗРОБКА МЕТОДІВ, МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ СИСТЕМИ

2.1 Розробка архітектури системи

Для розробки системи автоматизованого формування і редагування текстових документів було обрано архітектурний патерн MVVM.

Model-View-ViewModel (MVVM) [9] – архітектурний патерн проектування, який забезпечує чіткий розподіл компонентів системи, що сприяє підвищенню модульності різних її складових, зручнішому тестуванню та зручнішому обслуговуваню. Розподіл різних складових, таких як бізнес-логіка та інтерфейс, сприяє кращій масштабованості системи, дозволяє легше розширювати функціонал застосунку, сприяє написанню зрозумілішого коду, полегшує тестування окремих її компонентів та прискорює виявлення джерела можливих помилок.

Model (модель) представляє структури даних і бізнес-логіку програми. Для редактора текстових документів модель містить моделі даних для документів, розділів, абзаців, стилів і правил форматування, а також логіку для маніпулювання текстом і перевірки. Також, ця частина містить функції та алгоритми створення, редагування, форматування та збереження документів. Це включає логіку для автоматизованих завдань, як-от застосування шаблонів, виконання правил форматування або виконання маніпуляцій з текстом, таких як операції пошуку та заміни.

View (інтерфейс користувача) відповідає за безпосередню взаємодію з користувачем, відображаючи дані та фіксуючи введені користувачем дані.

ViewModel діє як посередник між моделю та інтерфейсом користувача. Вона обробляє логіку представлення даних та керування станом, надає потоки даних, обробляє взаємодії користувача та за потреби оновлює інтерфейс. Це забезпечує узгоджену поведінку та логіку на всіх платформах. Ця складова архітектури може містити функції і команди, які View може викликати для взаємодії з бізнес-логікою програми. Крім цього, ViewModel може відповідати

за мезанізми перевірки відповідності вимогам оброблюваних даних. Тут же можна реалізувати логіку для повідомлення користувача про помилки або запису таких ситуацій у логи системи для їх вивчення та виправлення можливих помилок при роботі програми у майбутньому.

Розроблена та описана архітектура системи наведена на рисунку 2.1.

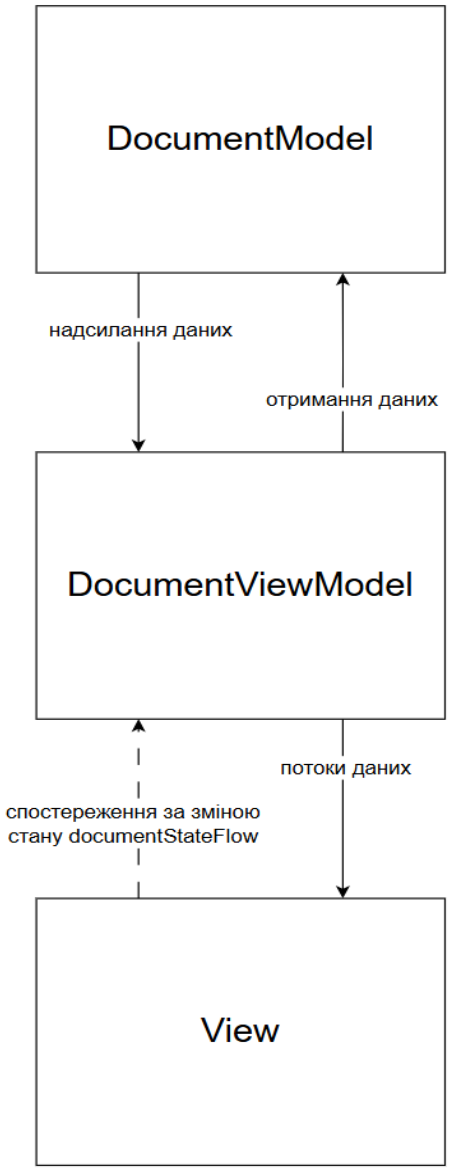

Рисунок 2.1 – Архітектура системи

Традиційний підхід до розробки системи передбачає, що об’єкт самостійно відповідає за створення та управління своїми залежностями, що призводить до жорсткої зв’язності компонентів, що ускладнює їх зміну та

тестування.

Для покращення моделі системи, зменшення зв’язності між компонентами, підвищення гнучкості системи та полегшеного тестування, прийнято рішення застосувати в архітектурі системи шаблон проектування «впровадження залежностей» (Dependency Injection) [10]. Це дозволяє вирішити проблему із жорсткою зв’язністю компонентів шляхом передачі необхідних залежностей через конструктор або методи. Таким чином, наприклад, компонент View не створює екземпляр ViewModel, що може призвести до неправильної його роботи тоді, коли ViewModel використовують кілька View, а отримує його із зовнішнього джерела, що покращує архітектуру системи.

Приклад архітектури з використанням впровадження залежностей наведено на рисунку 2.2. Так, MainViewModel отримує доступ до AI Client та Document DAO, які використовує у своїй роботі, через конструктор з анотацією @Inject, а інтерфейс користувача (UI) отримує доступ до MainViewModel за допомогою функції hiltViewModel(). Для цього, необхідно додати анотацію @HiltViewModel до MainViewModel [11].

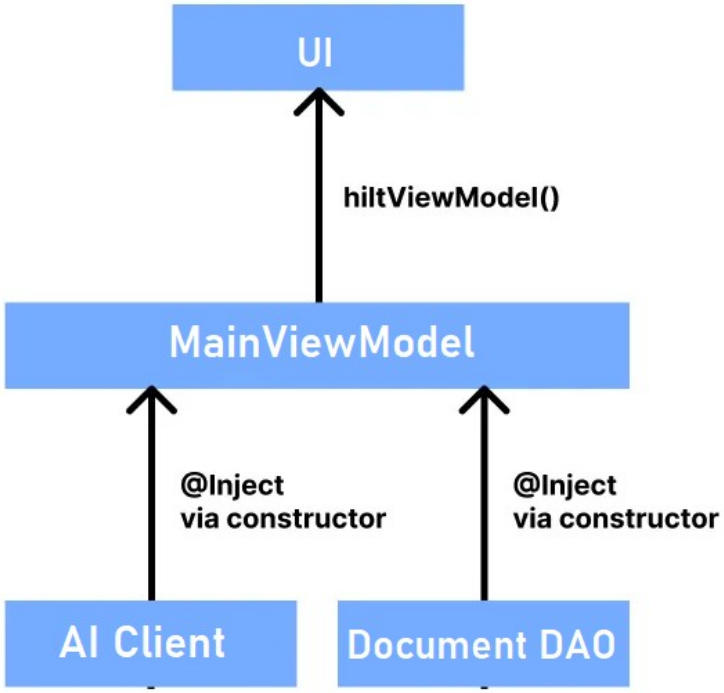

Рисунок 2.2 – Приклад архітектури з використанням впровадження залежностей

Використання архітектури MVVM покращує тестування та підтримки розроблюваного коду. Кожна зі складових системи може бути протестована незалежно одна від одної, що дозволяє швидше виявити та усунути можливу причину помилкової роботи системи [12]. Також, такий підхід дозволяє легше розширювати функціонал системи, додавати нові функції, вносити зміни до логіки роботи наявних функцій, оновлювати інтерфейс користувача.

2.2 Розробка методу формування структури документа

Цей метод дозволяє користувачам генерувати спеціальну структуру документа на основі типу документа, який вони створюють, і їхнього конкретного введення. На відміну від статичних шаблонів, цей метод адаптується в режимі реального часу до потреб користувача, створюючи динамічну структуру, адаптовану до мети та аудиторії документа.

Виконання методу складається з таких кроків:

1. Введення теми та призначення документу користувачем у діалоговому вікні, що реалізується класом Dialog.

2. На основі цих вхідних даних викликається клас AIClient, який надсилає запит штучному інтелекту на формування структури документа, що відповідає вказаній темі та призначенню документа.

3. AIClient повертає відповідь у форматі JSON, яка отримується системою за допомогою класу HttpClient та конвертується класом Json у формат, який можна вставити у текстовий редактор.

4. Система створює налаштовану структуру документа із відповідними розділами у текстовому редакторі, що реалізований з використанням бібліотеки Aztec.

5. Коли користувач заповнює вміст, система надає пропозиції щодо додаткових розділів або підрозділів, якщо виявлені прогалини в структурі документа.

Цей метод дозволяє частково автоматизувати процес структурування документа та більше зосереджуватися на створенні самого вмісту документа і

менше – на його правильному структуруванні і оформленні. Крім цього, система може запропонувати кращий спосіб оформлення того чи іншого документа, ніж той, який використовує користувач.

Блок-схема алгоритму роботи методу наведена на рисунку 2.3.

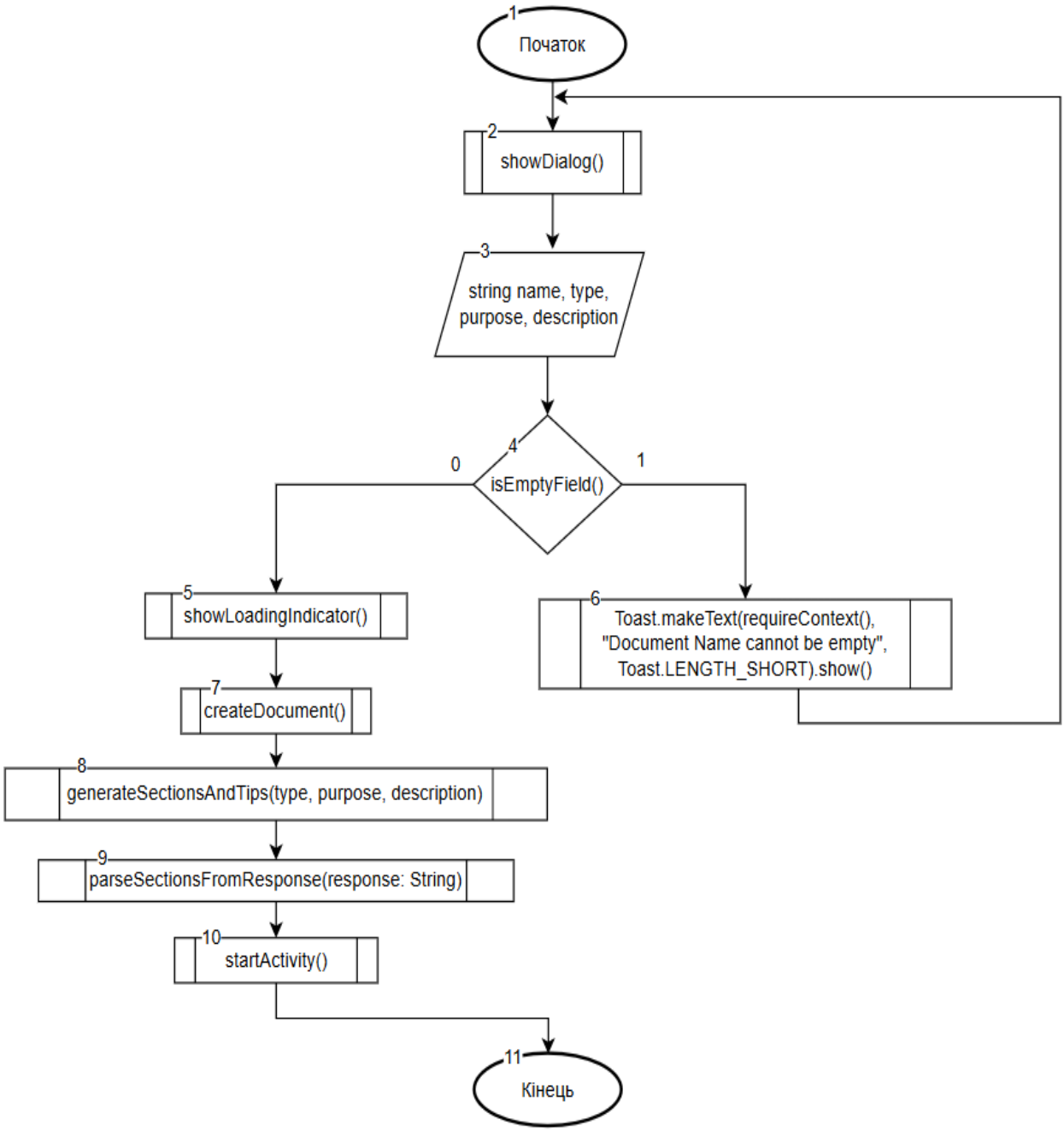

Рисунок 2.3 – Блок-схема алгоритму роботи методу формування структури документа

Для реалізації цього методу, розроблено класи, які описані в діаграмі

класів на рисунку 2.4.

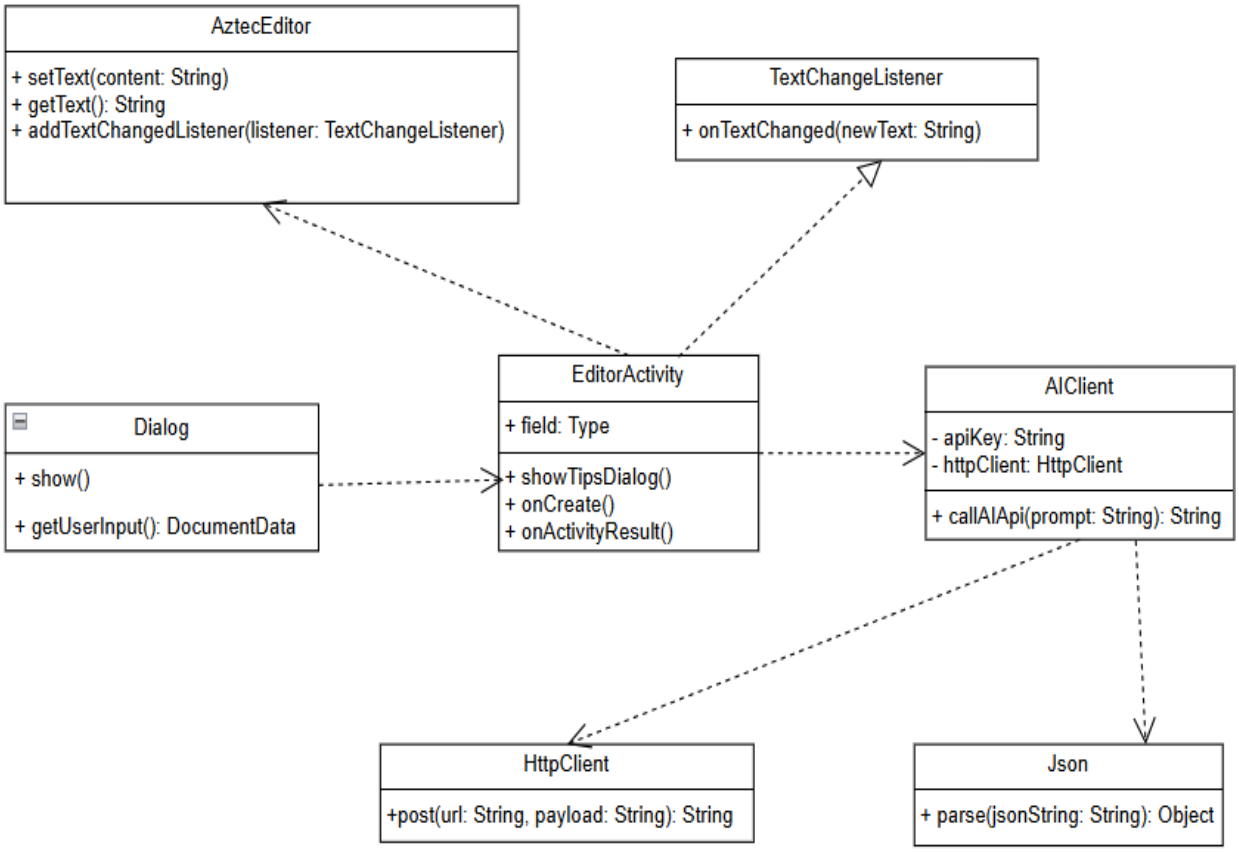

Рисунок 2.4 – Діаграма класів для методу автоматичної генерації структури документа

Клас Dialog забезпечує інтерфейс для введення користувачем теми та призначення документа. Для цього він використовує такі функції:

– show() – відображає діалогове вікно для введення даних;

– getUserInput(): DocumentData – отримує введені користувачем дані.

Клас EditorActivity – центральний клас, який координує роботу між іншими компонентами. Для цього він використовує такі функції:

– showTipsDialog() – викликається при натисканні на розділ документа для показу рекомендацій щодо наповнення документа;

– onCreate() – функція, що запускає текстовий редактор.

Клас AIClient відправляє запити до AI для генерації структури документа та отримання пропозицій. Він використовує функцію

callAiAPI(prompt:String):String, яка відправляє запит до штучного інтелекту.

Клас HttpClient відправляє HTTP-запити до AI сервера. Для цього використовується функція post(url: String, payload: String): String, яка відправляє POST-запит та повертає відповідь.

Клас Json парсить JSON-відповіді від AI сервера. Він використовує функцію parse(jsonString: String): Object, якапарсить JSON-рядок у об'єкт.

Клас AztecEditor – текстовий редактор для відображення та редагування документа. Для цього він використовує такі функції:

– setText(content: String): void – встановлює текст у редакторі.

– getText(): String – отримує поточний текст з редактора.

– addTextChangedListener(listener: TextChangeListener): void – додає слухача змін тексту.

Інтерфейс TextChangeListener – це слухач для відстеження змін у текстовому редакторі. В ньому використовується функція onTextChanged(newText: String): void, яка викликається при зміні тексту.

Для детального опису дій, що відбуваються під час виконання методу, побудовано таблицю 2.1.

Таблиця 2.1 – Повідомлення, якими обмінюються об’єкти

№	Відправник	Одержувач	Опис
1	Користувач	Dialog	Введення теми та призначення документа.
2	Dialog	EditorActivity	Передача введених даних для подальшої обробки.
3	EditorActivity	AIClient	Формування запиту та виклик методу generateDocumentStructure() для отримання структури документа.
4	AIClient	HttpClient	Формування HTTP-запиту для відправки до AI сервера.
5	HttpClient	AI Server	Відправка сформованого запиту для обробки.

Продовження таблиці 2.1

№	Відправник	Одержувач	Опис
6	AI Server	HttpClient	Обробка запиту та повернення відповіді у форматі JSON.
7	HttpClient	AIClient	Повернення отриманої відповіді.
8	AIClient	Json	Парсинг отриманої відповіді у структуру даних.
9	Json	AIClient	Повернення розпарсених структурованих даних.
10	AIClient	EditorActivity	Повернення структурованих даних (структури документа).
11	EditorActivity	AztecEditor	Відображення структури документа у текстовому редакторі.
12	Користувач	AztecEditor	Заповнення вмісту документа, використовуючи надану структуру.
13	AztecEditor	EditorActivity	Виклик методу onTextChanged() для повідомлення про зміни при зміні тексту.
14	AIClient	AI Server	Відправка запиту для рекомендацій.
15	AI Server	HttpClient	Повернення відповіді з рекомендаціями у форматі JSON.
16	HttpClient	AIClient	Передача відповіді.
17	AIClient	Json	Парсинг отриманої відповіді.
18	Json	AIClient	Повернення розпарсених рекомендацій.
19	AIClient	EditorActivity	Відображення отриманих рекомендацій.
20	Користувач	AztecEditor	Користувач ознайомлюється з пропозиціями та вирішує, чи додавати їх до документа.

На основі цього опису розроблено діаграму послідності, яка зображена на

рисунку 2.5.

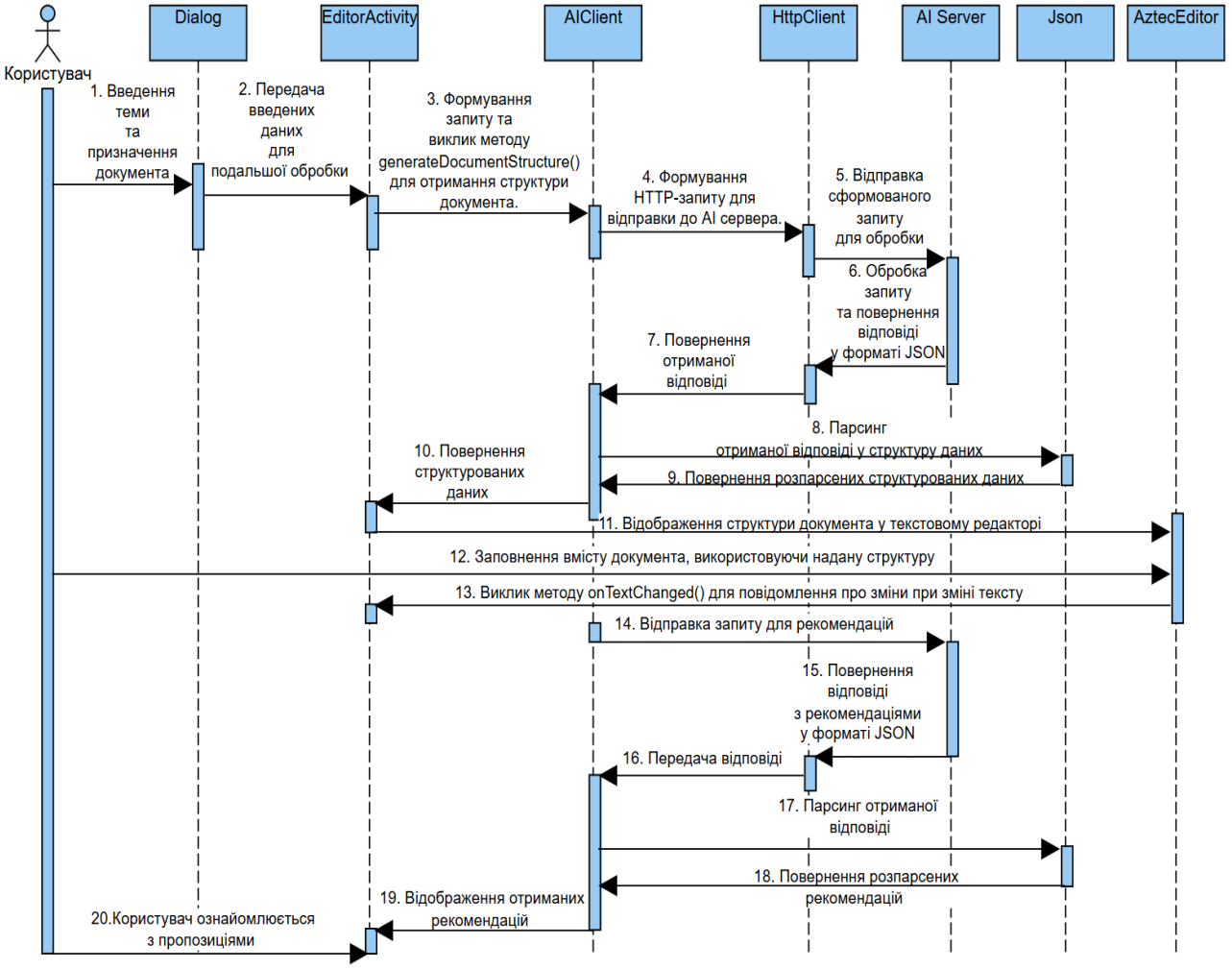

Рисунок 2.5 – Діаграма послідовності

2.3 Розробка методу формування рекомендацій щодо наповнення документа

Цей метод надає рекомендації користувачеві щодо наповнення документа вмістом залежно від його теми, призначення та структури, що була сформована методом налаштування динамічної структури документа.

Виконання методу складається з таких кроків:

1. Користувач відкриває існуючий або створює новий документ.

2. Система формує структуру документа.

3. Система надсилає запит до штучного інтелекту з використанням класу AIClient для надання рекомендацій щодо наповнення кожного розділу

документа змістом.

4. AIClient повертає відповідь у форматі JSON, яка отримується системою за допомогою класу HttpClient та конвертується класом Json.

5. Система прикріплює рекомендації до кожного розділу та створює клікабельні розділи з використанням класу SpannableString.

6. Користувач натискає на потрібний розділ та отримує рекомендації у вспливаючому вікні, що реалізовано з використанням класу Dialog.

Блок-схема алгоритму роботи цього методу наведена на рисунку 2.6.

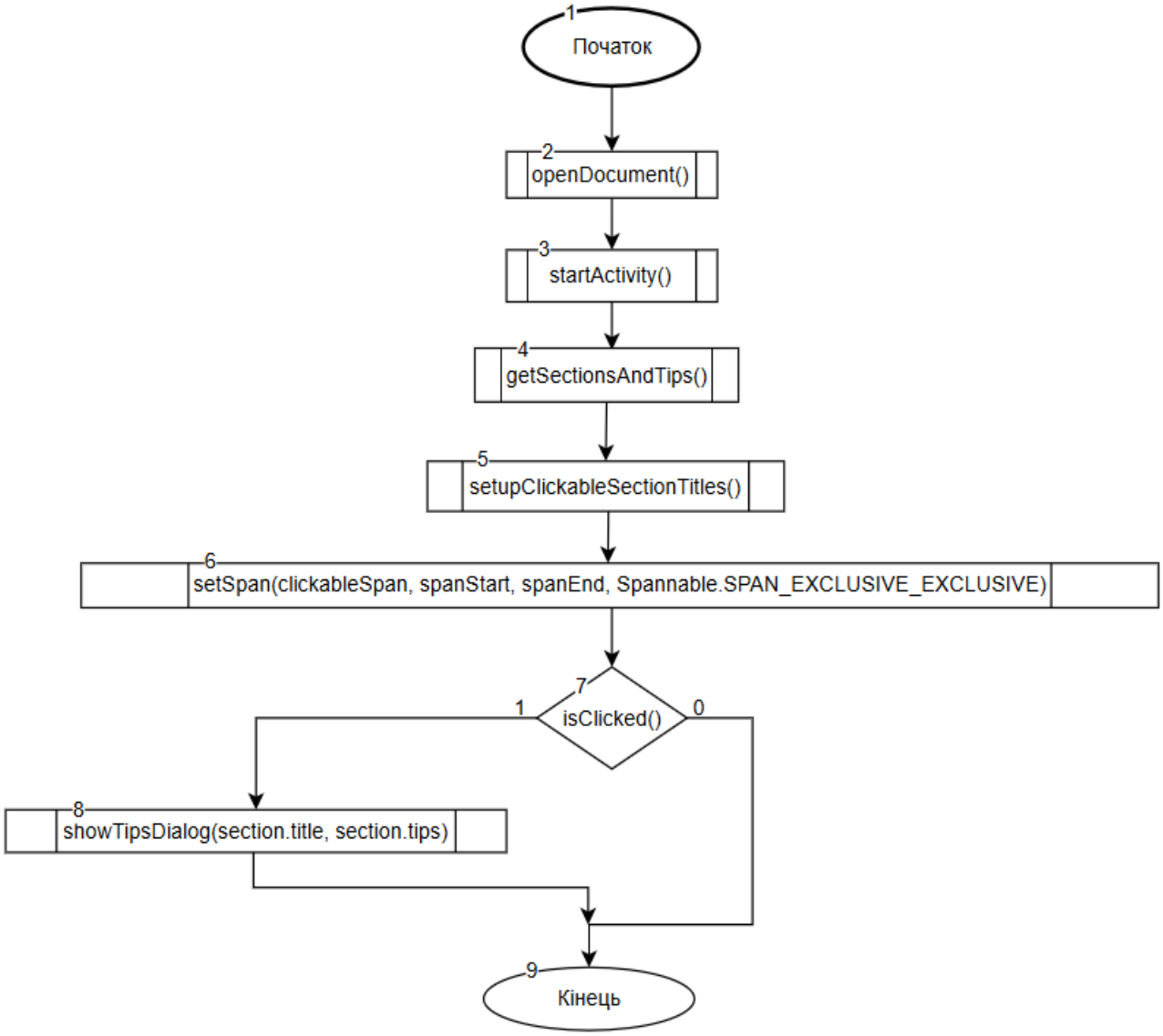

Рисунок 2.6 – Блок-схема алгоритму роботи

Для реалізації цього методу, розроблено класи, які описані в діаграмі

класів на рисунку 2.7.

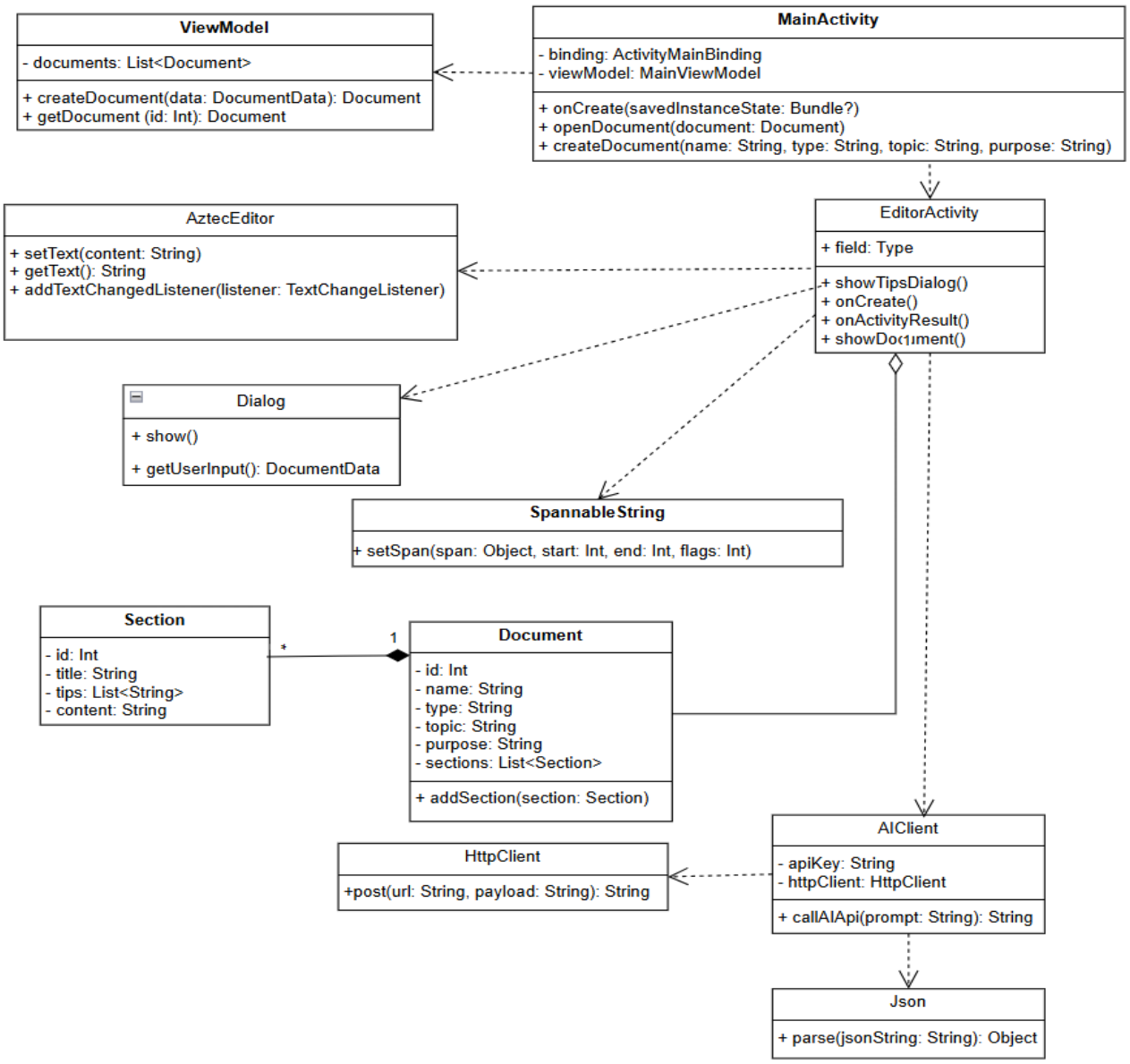

Рисунок 2.7 – Діаграма класів, що реалізують метод формування рекомендацій щодо наповнення документа

Клас MainActivity – головний клас застосунку, дозволяє користувачу відкрити існуючий або створити новий документ.

Для цього використовуються такі функції:

– onCreate(savedInstanceState: Bundle?) – ініціалізує інтерфейс та встановлює слухачів подій;

– openDocument(document: Document) – відкриває існуючий документ;

– createDocument(name: String, type: String, topic: String, purpose: String – ініціює створення нового документа.

Клас ViewModel управляє створенням, зберіганням та отриманням документів.

Для цього використовуються такі функції:

– createDocument(data: DocumentData): Document – Створює новий документ;

– getDocument(id: String): Document – Отримує існуючий документ за ідентифікатором.

Клас Document представляє документ із його структурою та вмістом.

Він містить такі атрибути:

– id: String – унікальний ідентифікатор документа;

– name: String – назва документа;

– type: String – тип документа;

– topic: String – тема документа;

– purpose: String – призначення документа;

– sections: List<Section> – список розділів документа.

Використовується функція addSection(section: Section), яка додає розділ до документа.

Клас Section представляє розділ документа. Він містить такі атрибути:

– id: Int – Ідентифікатор розділу.

– title: String – Назва розділу.

– tips: List<String> – Рекомендації щодо вмісту розділу.

– content: String – Вміст розділу.

Клас SpannableString створює текст з можливістю додавання спанів (наприклад, клікабельних). Для цього використовується функція setSpan(span: Object, start: Int, end: Int, flags: Int), яка встановлює спан на текст.

Для детального опису дій, що відбуваються під час виконання методу, побудовано таблицю 2.2.

Таблиця 2.2 – Обмін подіями та повідомленнями між класами

№	Відправник	Одержувач	Опис
1	Користувач	MainActivity	Відкривання існуючого документа або створення нового.
2	MainActivity	DocumentManager	Отримання або створення документа.
3	DocumentManager	Document	Створення об’єкта документ.
4	MainActivity	EditorActivity	Запуск EditorActivity з передачею даних.
5	EditorActivity	EditorActivity	Ініціалізація запуску редактора.
6	EditorActivity	AztecEditor	Відображення документа у редакторі.
7	EditorActivity	AIClient	Запит на отримання рекомендацій для кожного розділу.
8	AIClient	HttpClient	Відправка HTTP POST-запиту до AI сервера.
9	HttpClient	AI Server	Отримання та обробка запиту.
10	AI Server	HttpClient	Повернення рекомендації у форматі JSON.
11	HttpClient	AIClient	Передача відповіді.
12	AIClient	Json	Парсинг JSON-відповіді.
13	Json	AIClient	Повертення розпарсених рекомендацій.
14	AIClient	EditorActivity	Передача рекомендацій.
15	EditorActivity	SpannableString	Створення клікабельних розділів з прикріпленими рекомендаціями.
16	Користувач	EditorActivity	Натискання на клікабельний розділ у редакторі.
17	EditorActivity	Dialog	Відображення діалогового вікна з рекомендаціями для вибраного розділу.

Таким чином, було розроблено метод формування рекомендацій щодо наповнення документа, побудовано класи, діаграму класів для його реалізації, описано повідомлення та дії, якими вони обмінюються під час виконання, побудовано діаграму послідовності, побудовано блок-схему алгоритму роботи цього методу.

2.4 Розробка методу рекомендацій елементів візуалізації

Метод рекомендацій елементів візуалізації пришвидшує користувачам процес вибору та побудови елементів візуалізації у їхніх документах (за потреби), надає найбільш релевантні рекомендації згідно контексту. Це дозволяє сформувати більш інформативні документи та прискорити процес їх наповнення інтерактивним змістом.

Виконання методу складається з таких кроків:

1. Користувач створює новий документ з використанням діалогового вікна Dialog.

2. Користувач формує зміст певної частини документа у текстовому редакторі.

3. Система аналізує в режимі реального часу зміну документа з використанням методу textChangeObserver() або ж користувач сам виділяє потрібну частину тексту та викликає вбудованого у застосунок помічника.

4. Викликається метод recommendInteractiveElements(), який з допомогою класу AIClient надсилає запит штучному інтелекту на отримання рекомендацій по додаванню елементів візуалізації.

5. На основі отриманої відповіді, створюється елемент візуалізації з використанням бібліотеки MPAndroidChart (якщо це графік або діаграма) або методу buildTable (якщо це таблиця).

6. Створений елемент візуалізації додається до документа з використанням класів Bitmap та Canvas.

Блок-схему алгоритму роботи цього методу наведено на рисунку 2.8.

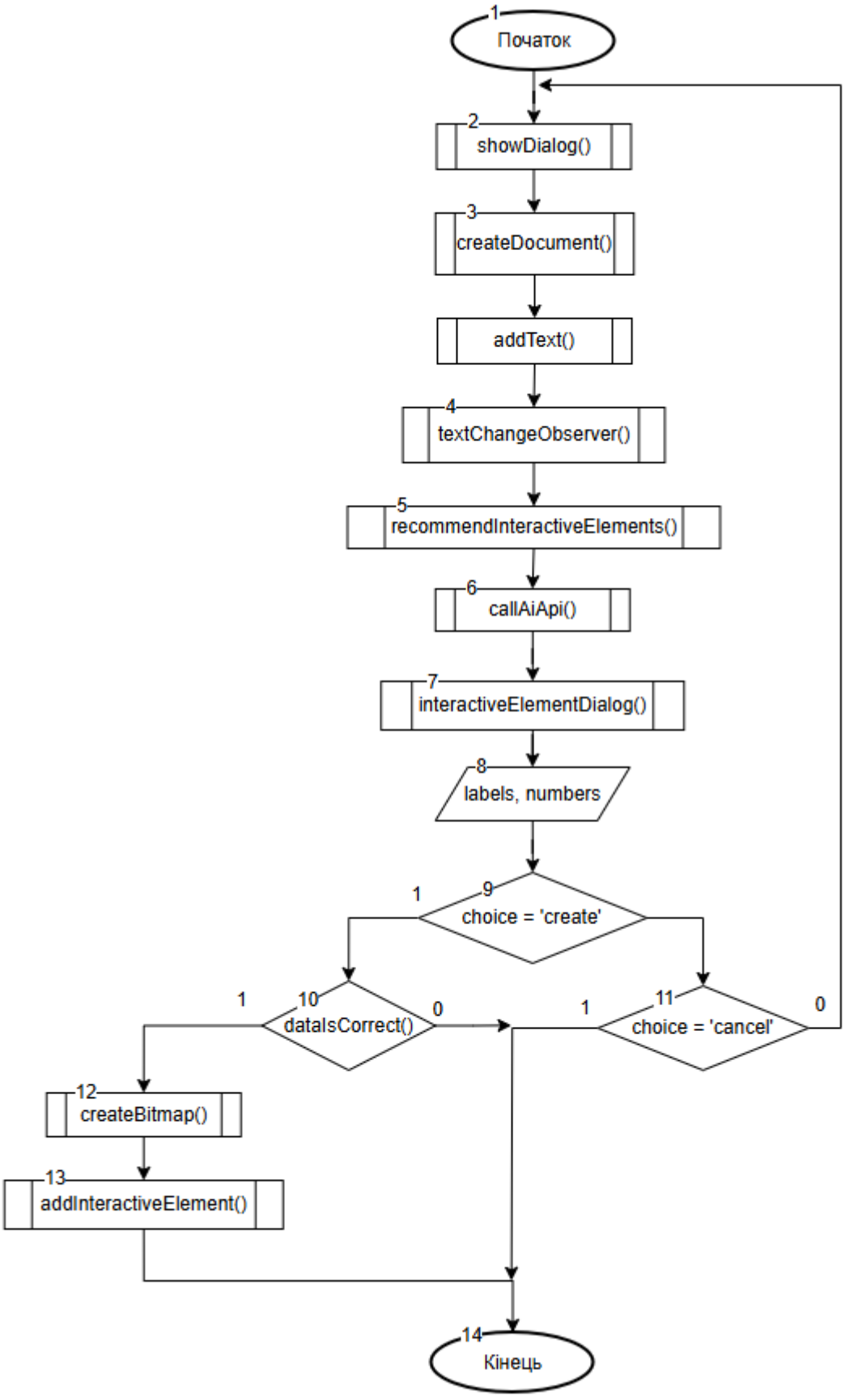

Рисунок 2.8 — Блок-схема алгоритму роботи методу рекомендацій елементів візуалізації

Для реалізації цього методу, розроблено класи, які описані в діаграмі класів на рисунку 2.9.

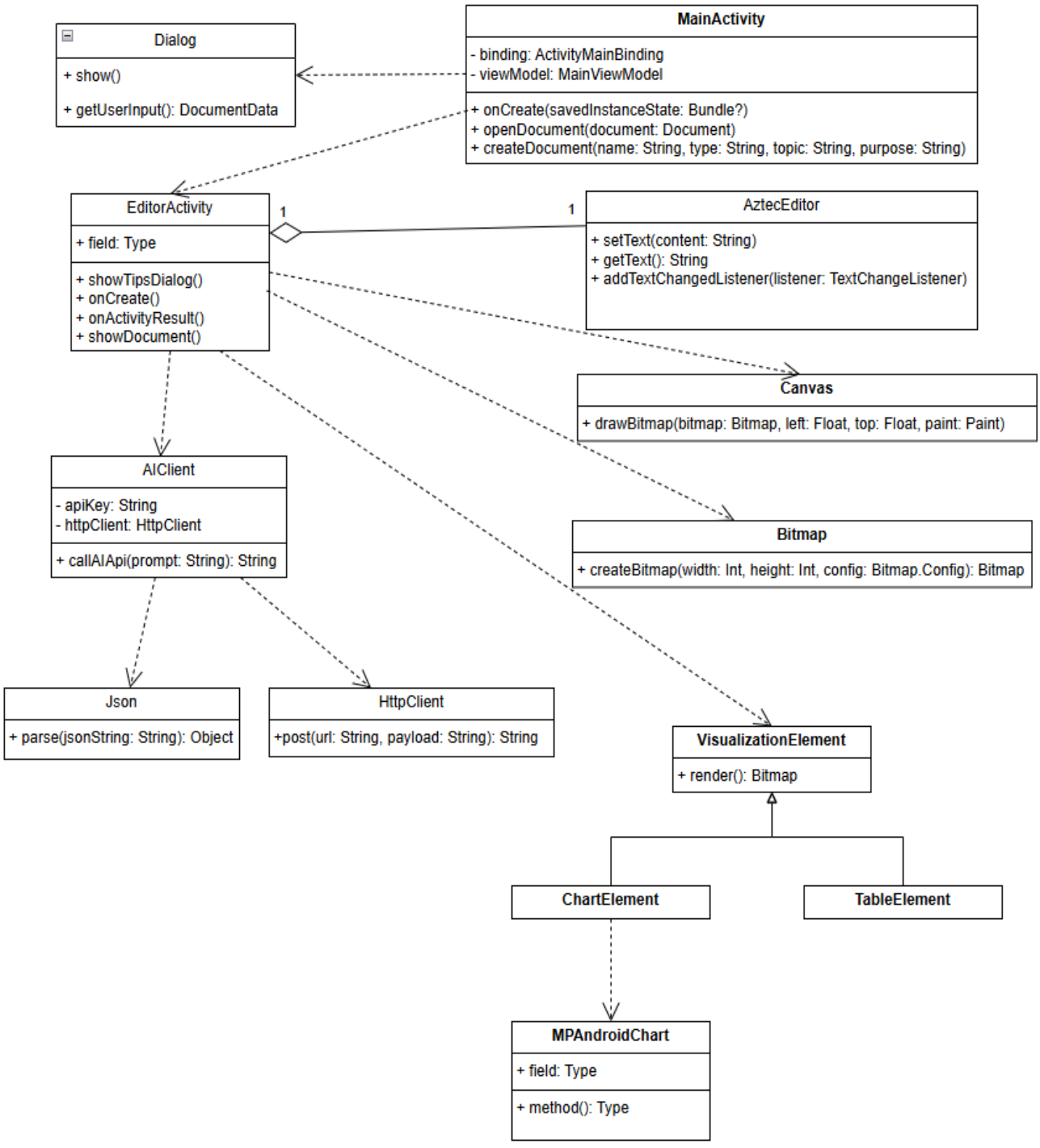

Рисунок 2.9 – Діаграма класів, що реалізують метод рекомендацій елементів візуалізації

Клас VisualizationElement – абстрактний клас для різних типів елементів

візуалізації. Для своєї роботи він використовує функцію render(): Bitmap, яка повертає зображення елемента. Цей клас містить підкласи: ChartElement – для графіків та діаграм, TableElement – для таблиць.

Клас MPAndroidChart – бібліотека для створення графіків та діаграм. Для своєї роботи він використовує функцію createChart(data: ChartData): Chart, яка створює графік на основі даних.

Клас Bitmap та Canvas використовуються для роботи з зображеннями та їх відображення на екрані.

Для цього використовуються такі методи:

– Bitmap.createBitmap(width: Int, height: Int, config: Bitmap.Config): Bitmap – створює нове зображення;

– Canvas.drawBitmap(bitmap: Bitmap, left: Float, top: Float, paint: Paint) – малює зображення на канві.

Для детального опису дій, що відбуваються під час виконання методу, побудовано таблицю 2.3.

Таблиця 2.3 – Повідомлення, якими обмінюються об’єкти

№	Відправник	Одержувач	Опис
1	Користувач	MainActivity	Ініціація створення нового документа.
2	MainActivity	Dialog	Відображення діалогового вікна для введення даних.
3	Користувач	Dialog	Введення необхідних даних.
4	Dialog	MainActivity	Передача введених даних.
5	MainActivity	EditorActivity	Запуск EditorActivity для редагування документа.
6	EditorActivity	AztecEditor	Ініціалізація порожнього документа у редакторі.
7	Користувач	AztecEditor	Редагування тексту документа.
8	AztecEditor	EditorActivity	Повідомлення про зміну тексту.

Продовження таблиці 2.3

№	Відправник	Одержувач	Опис
9	EditorActivity	EditorActivity	Аналіз змін в тексті в режимі реального часу.
10	Користувач	AztecEditor	Виділення частини тексту.
11	AztecEditor	EditorActivity	Повідомлення про виділення тексту.
12	EditorActivity	EditorActivity	Виклик методу для отримання рекомендацій.
13	EditorActivity	AIClient	Відправлення запиту для отримання рекомендацій.
14	AIClient	HttpClient	Відправлення запиту до AI Server.
15	HttpClient	AI Server	Отримання та обробка запиту.
16	AI Server	HttpClient	Повернення рекомендацій у форматі JSON.
17	HttpClient	AIClient	Передача відповіді.
18	AIClient	Json	Парсинг JSON-відповіді.
19	Json	AIClient	Повернення розпарсених рекомендацій.
20	AIClient	EditorActivity	Передача рекомендацій.
21	EditorActivity	VisualizationElement	Створення елементу візуалізації на основі рекомендацій.
22	VisualizationElement	MPAndroidChart/BuildTable	Створення графіку, діаграми або таблицю.
23	VisualizationElement	EditorActivity	Повертення зображення елемента.
24	EditorActivity	AztecEditor	Додавання елемента візуалізації в документ.

Таким чином, було розроблено метод рекомендацій елементів візуалізації, побудовано класи, діаграму класів для його реалізації, описано повідомлення та дії, якими вони обмінюються під час виконання.

2.5 Розробка методу роботи з елементами візуалізації у Android View інтерфейсі

Метод роботи з елементами візуалізації у Android View інтерфейсі дозволяє працювати з таблицями та діаграмами безпосередньо у редакторі тексту, що не має прямої реалізації.

Метод складається з таких кроків:

1. Користувач обирає елемент візуалізації для додавання у спеціальному меню.

2. З’являється діалогове вікно Dialog для взаємодії з користувачем і отримання даних для створення елемента візуалізації.

3. Користувач вводить необхідні дані у спеціально відведені поля.

4. Система перетворює елемент візуалізації у зображення з використанням методу convertToBitmap().

5. Система викликає метод insertImage(), який працює за допомогою бібліотеки Glide для роботи із зображеннями.

6. Створене зображення елемента візуалізації додається до текстового редактора.

Цей метод створює елементи візуалізації у вигляді зображень, при цьому при натисканні на них повторно з’являється діалогове вікно з раніше уведеними даними та надає можливість внести зміни. Після внесених змін зображення заміниться на оновлене.

Для реалізації цього методу, розроблено класи, які описані в діаграмі класів на рисунку 2.10.

Клас VisualizationMenu відображає спеціальне меню з доступними елементами візуалізації.

Для цього використовуються такі функції:

– showMenu() – відображає меню елементів візуалізації;

– setOnElementSelectedListener(listener: OnElementSelectedListener) – встановлює слухача для обробки вибору елемента.

Клас VisualizationDialog представляє діалогове вікно для взаємодії з

користувачем та отримання необхідних даних для створення елемента візуалізації.

Для цього використовуються такі функції:

– show() – відображає діалогове вікно;

– getUserInput(): VisualizationData – отримує введені користувачем дані.

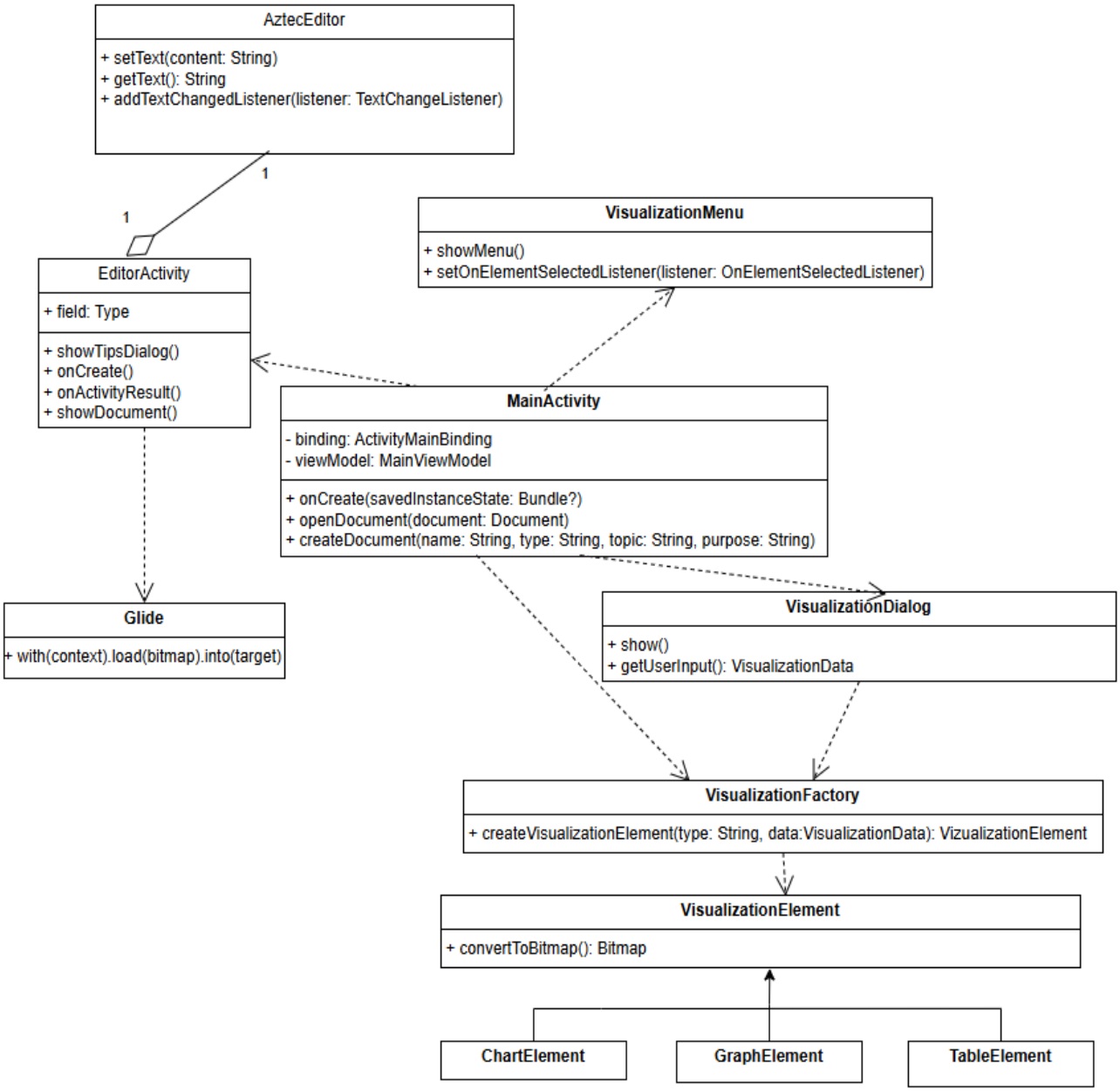

Рисунок 2.10 – Діаграма класів, що реалізують метод роботи з елементами візуалізації у Android View інтерфейсі

Клас VisualizationFactory створює об'єкти елементів візуалізації на основі

типу та даних. Для цього використовується функція createVisualizationElement(type: String, data: VisualizationData): VisualizationElement, яка створює елемент візуалізації.

Абстрактний клас VisualizationElement – базовий клас для всіх елементів візуалізації. Він використовує функцію convertToBitmap(): Bitmap, яка перетворює елемент візуалізації у зображення.

Підкласи: ChartElement, GraphElement, TableElement – конкретні реалізації елементів візуалізації.

Клас ChartElement – клас, який представляє реалізацію елемента візуалізації для графіків. Він використовує функцію convertToBitmap(): Bitmap, яка реалізує перетворення графіка у зображення, використовує бібліотеку MPAndroidChart для побудови графіка.

Клас Glide – бібліотека для роботи із завантаженням та відображенням зображень. Він використовує функцію with(context).load(bitmap).into(target) – завантажує зображення та вставляє його у вказаний елемент.

Для детального опису дій, що відбуваються під час виконання методу, побудовано таблицю 2.4.

Таблиця 2.4 – Повідомлення, якими обмінюються об’єкти

№	Відправник	Одержувач	Опис
1	Користувач	EditorActivity	Натискання на меню елементів візуалізації.
2	EditorActivity	VisualizationMenu	Відображення спеціального меню з елементами візуалізації.
3	VisualizationMenu	Користувач	Вибір типу елемента візуалізації.
4	VisualizationMenu	EditorActivity	Повідомлення про вибір елемента.
5	EditorActivity	Користувач	Відображення діалогового вікна для введення даних.
6	Користувач	VisualizationDialog	Заповнення поля з даними для елемента.

Продовження таблиці 2.4

№	Відправник	Одержувач	Опис
7	VisualizationFactory	VisualizationElement	Створення об'єкту елемента візуалізації відповідного типу.
8	VisualizationElement	VisualizationElement	Перетворення елементу візуалізації у зображення.
9	EditorActivity	EditorActivity	Виклик методу для вставки зображення у текстовий редактор.
10	EditorActivity	Glide	Використання Glide для вставки зображення.
11	Glide	AztecEditor	Вставлення зображення у текстовий редактор.

На основі цього опису, було побудовану діаграму послідності, яка зображена на рисунку 2.11.

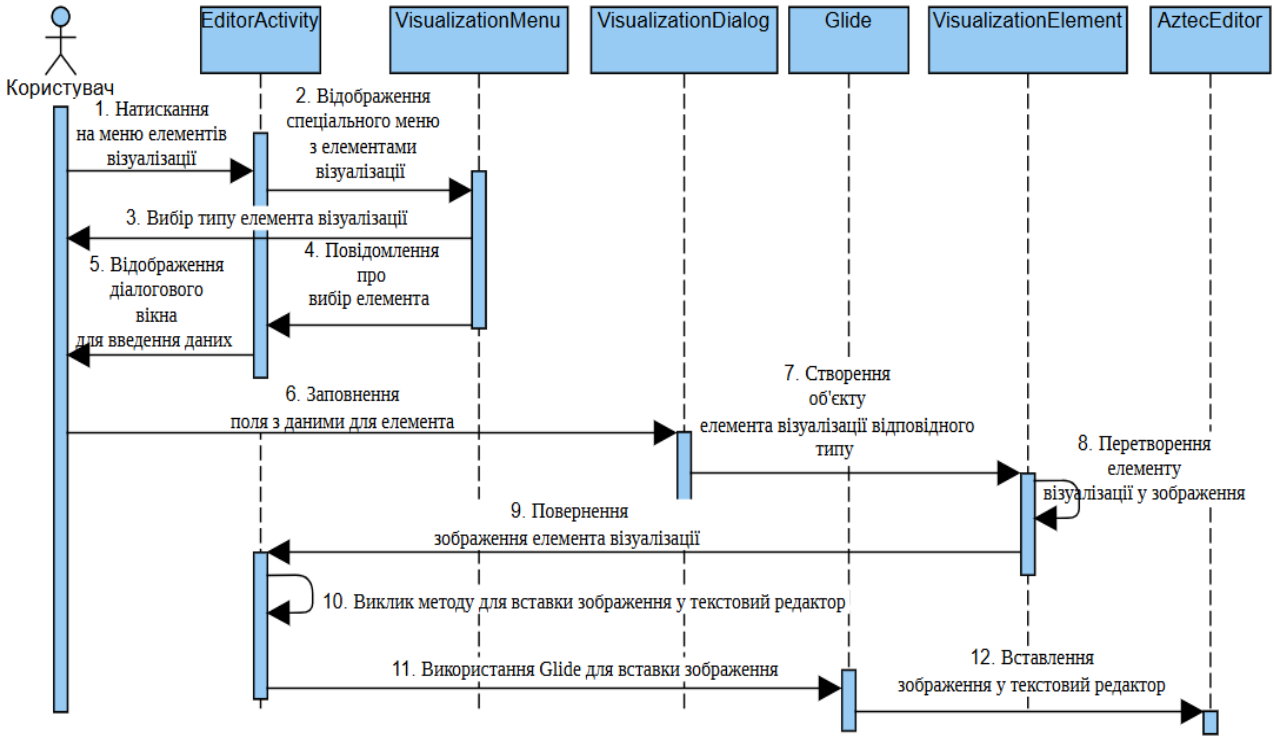

Рисунок 2.11 – Діаграма послідовності для методу роботи з елементами візуалізації у Android View інтерфейсі

Таким чином, було розроблено метод роботи з елементами візуалізації у Android View інтерфейсі, побудовано класи, діаграму класів для його реалізації, описано повідомлення та дії, якими вони обмінюються під час виконання, побудовано діаграму послідовності.

2.6 Розробка моделі та загального алгоритму роботи системи

На рисунку 2.12 зображено модель системи автоматизованого формування і редагування текстових документів, яка створена у вигляді UML діаграми варіантів використання.

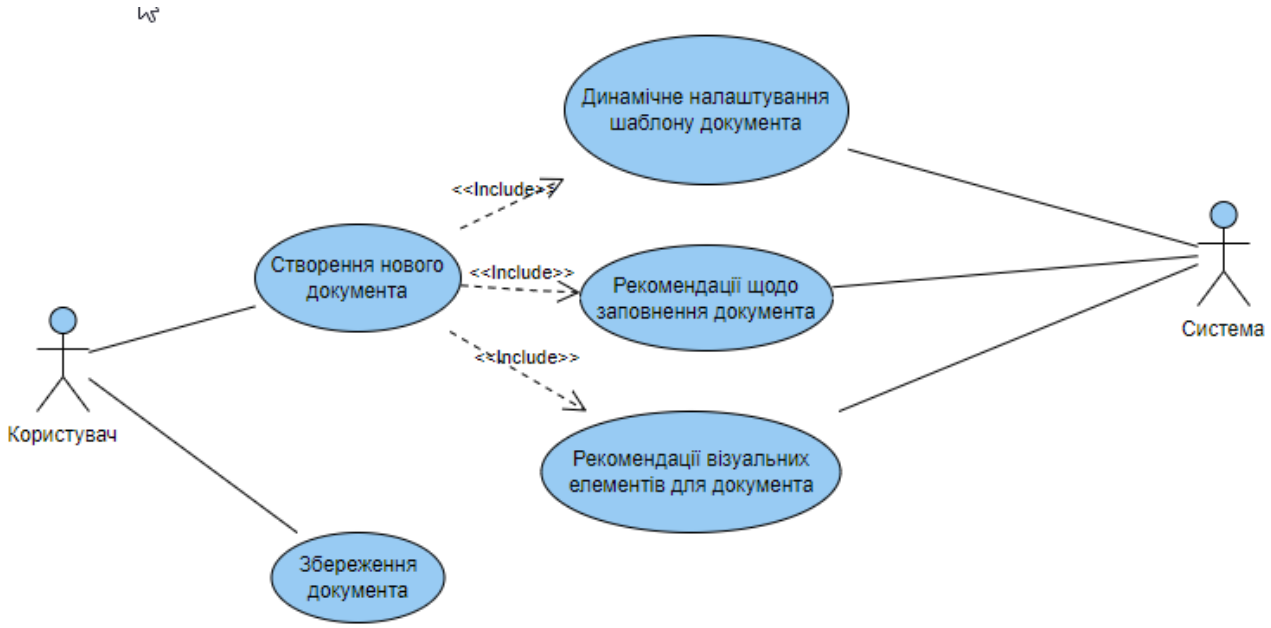

Рисунок 2.12 – Модель системи

Модель включає у себе 5 прецедентів: створення нового документа, збереження документа, динамічне налаштування шаблону документа, рекомендації щодо заповнення документа, рекомендації візуальних елементів для документа.

Потік подій для прецеденту «Динамічне налаштування структури документа».

Актор – користувач, система.

Мета – автоматично створити структуру документа на основі введених

даних користувача та типу документа.

Передумови – користувач має бути авторизованим у додатку., вказаний тип документа.

Результат – динамічна структура створюється на основі типу документа та потреб користувача.

Основний потік:

Користувач обирає тип документа зі списку (наприклад, звіт, бізнес-пропозиція).

Система запитує у користувача додаткову інформацію.

Система генерує початкову структуру документа.

Користувач може налаштувати структуру, додаючи або видаляючи розділи.

Користувач підтверджує структуру, і система зберігає її.

Альтернативні потоки:

Е-1: Відсутній тип документа – якщо користувач не обрав тип документа, система просить його вибрати один перед продовженням.

Е-2: Некоректний ввід – якщо користувач надає некоректні дані (наприклад, непідтримуваний тип документа), система виводить помилку і пропонує дійсні варіанти.

Потік подій для прецеденту «Збереження документа».

Актор: Користувач.

Мета: Зберегти поточний документ і експортувати його в різні формати.

Передумови:

1. Користувач має працювати з відкритим документом.

2. У документі повинно бути додано вміст.

Результат – документ зберігається і, за потреби, експортується у вибраний формат (наприклад, PDF, DOCX).

Основний потік:

1. Користувач обирає опцію «Зберегти».

2. Система автоматично зберігає документ у власному форматі програми.

3. Користувач обирає опцію Експорт і вибирає формат (наприклад, PDF, DOCX).

4. Система конвертує документ у вибраний формат і пропонує завантажити або зберегти його.

5. Користувач підтверджує, і система експортує файл у вибране місце.

Альтернативні потоки:

Е-1: Помилка збереження. Якщо система не може зберегти документ (наприклад, через недостатній обсяг пам’яті), користувач отримує сповіщення та пропонується звільнити місце або вибрати інше місце для збереження.

Е-2: Непідтримуваний формат експорту. Якщо користувач вибирає формат, який не підтримується системою, відображається повідомлення про помилку з альтернативними варіантами форматів.

Потік подій для прецеденту «Створення нового документа».

Актор: Користувач.

Мета: Створити новий документ із початковою структурою, відповідно до потреб користувача.

Передумови – користувач має бути авторизованим у системі. Результат – створений новий документ з обраною структурою та збережений у системі.

Основний потік:

1. Користувач обирає опцію Створити новий документ у меню. Система запитує у користувача тип документа (наприклад, звіт, пропозиція, презентація).

2. Користувач вводить назву документа та інші базові дані (наприклад, цільова аудиторія, тема).

3. Система автоматично генерує початкову структуру документа на основі вибраного типу.

4. Користувач може переглянути запропоновану структуру та внести зміни (додавати, видаляти або перейменовувати розділи).

5. Після підтвердження структури система створює новий документ і

зберігає його як чернетку.

6. Користувач починає редагувати документ, додаючи контент у відповідні розділи.

Альтернативні потоки:

Е-1. Відсутня назва документа. Якщо користувач не вказав назву документа, система відображає повідомлення про необхідність її введення перед продовженням.

Е-2. Вибрано невідомий тип документа. Якщо користувач вибирає тип документа, який не підтримується системою, відображається повідомлення про помилку, і пропонуються доступні варіанти типів документів.

Е-3. Відмова від створення. Користувач може на будь-якому етапі скасувати створення документа, після чого система повертає його до головного меню без збереження змін.

Потік подій для прецеденту «Рекомендації щодо вмісту заповнення документа».

Актор – користувач, система.

Мета – отримати рекомендації щодо того, що слід включити до кожного розділу документа на основі його типу і теми.

Передумови – користувач уже створив або вибрав структуру документа, Користувач вказав тип і тему документа.

Результат – система надає рекомендації щодо вмісту для кожного розділу документа.

Основний потік:

1. Користувач вибирає розділ документа.

2. Система аналізує тип і тему документа.

3. Система надає рекомендації щодо вмісту, специфічні для цього розділу (наприклад, додати ключові фінансові показники, аналіз ринку).

4. Користувач приймає або змінює рекомендації та додає вміст до розділу.

5. Система зберігає документ з оновленим вмістом.

Було також розроблено блок-схему загального алгоритму роботи системи,

яка демонструє усю можливу взаємодію користувача із системою. Блок-схема наведена на рисунку 2.13.

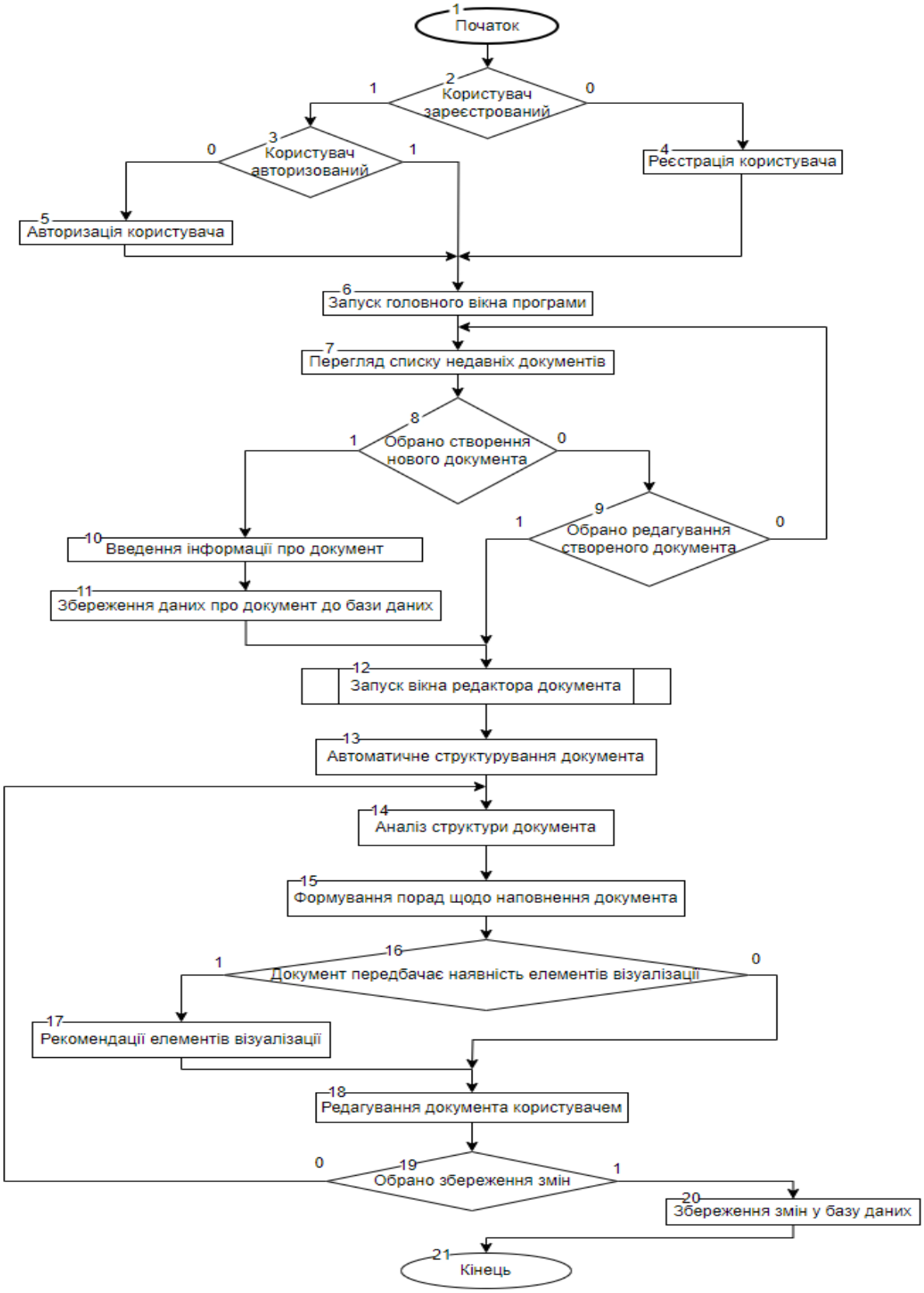

Рисунок 2.13 – Блок-схема загального алгоритму роботи системи

2.7 Розробка структури класів для роботи з даними

Для структурування даних, які генеруються та використовуються під час роботи програми, для збереження файлів з метою їх подальшого редагування й використання, включно з усім текстом та елементами візуалізації, розподілом папок на файли, розроблено структуру класів даних.

Структуру класів даних наведено на рисунку 2.14.

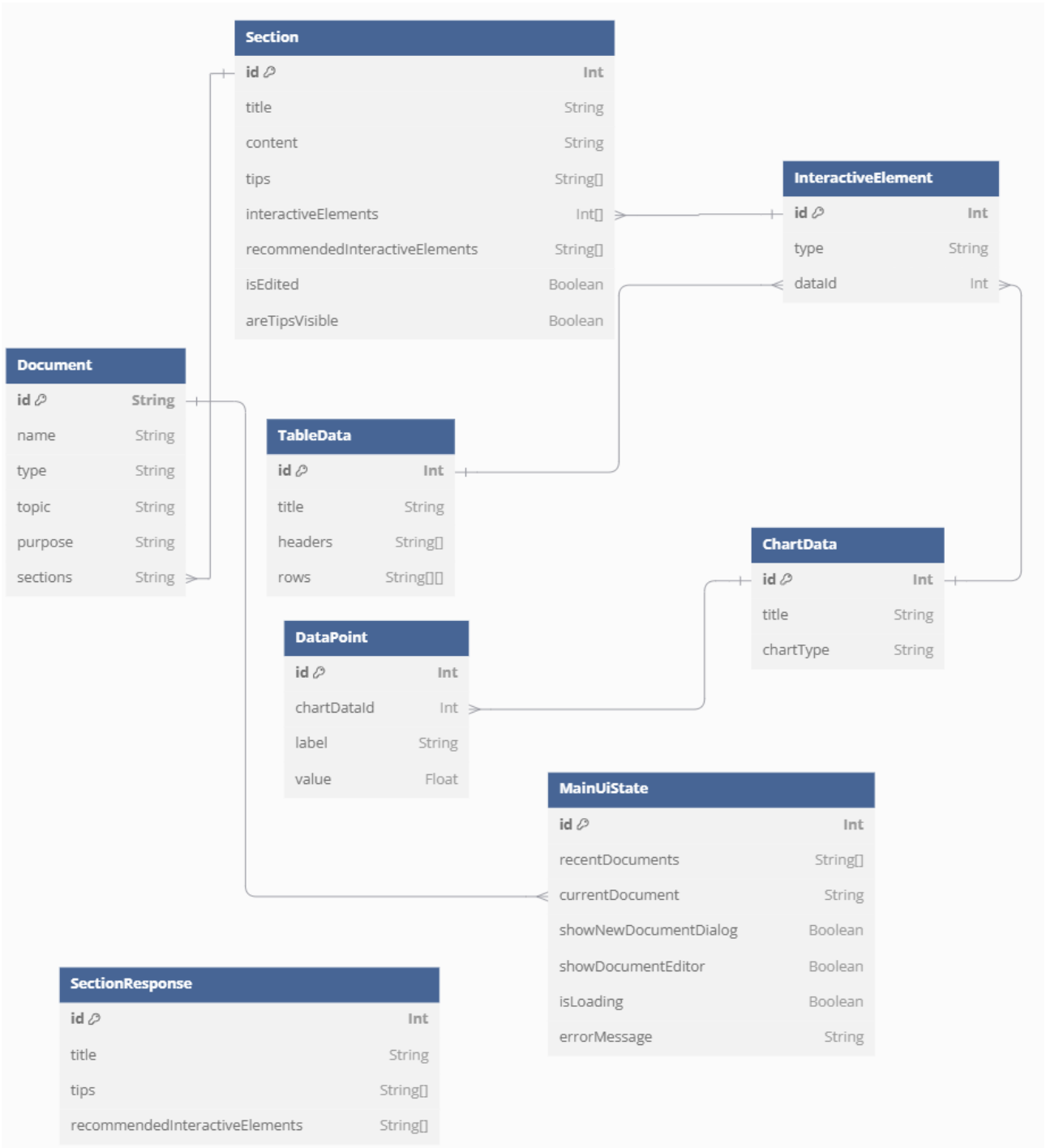

Рисунок 2.14 – Структура класів даних

Клас Document містить інформацію про документи. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного документа (String);

– name – назва документа (String);

– type – тип або категорія документа (String);

– topic – основна тема або предмет документа (String);

– purpose – призначення або мета документа (String);

– sections – посилання на розділи, що є частиною цього документа (список ідентифікаторів Section).

Клас Section містить деталі про розділи всередині документа. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного розділу (Int);

– title – назва або заголовок розділу (String);

– content – основний вміст або текст розділу (String);

– tips – список корисних порад, пов'язаних із розділом (Список String);

– interactiveElements – посилання на інтерактивні елементи, що містяться в розділі (Список ідентифікаторів InteractiveElement);

– recommendedInteractiveElements – рекомендовані типи інтерактивних елементів для розділу (список String);

– isEdited – булеве значення, яке вказує, чи був розділ відредагований (Boolean);

– areTipsVisible – булеве значення, яке вказує, чи повинні бути видимими поради у розділі (Boolean).

Клас InteractiveElement представляє різні типи інтерактивних елементів, які можна вбудувати в розділи. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного інтерактивного елемента (Int);

– type – тип інтерактивного елемента (наприклад, Chart, Table, Graph) (String);

– data – посилання на дані, пов'язані з цим інтерактивним елементом, такі як дані діаграми, таблиці або графа (посилання на ChartData, TableData або GraphData).

Клас ChartData містить дані для діаграм, що використовуються як інтерактивні елементи. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного запису даних діаграми (Int);

– title – назва або заголовок діаграми (String);

– chartType – тип діаграми (String).

Таблиця DataPoint таблиця зберігає окремі точки даних, що належать до певної діаграми. Вона містить такі поля:

– id – унікальний ідентифікатор для кожної точки даних (Int);

– chartDataId – посилання на діаграму, до якої належить ця точка даних (Int);

– label – иітка для точки даних, що зазвичай описує, що вона представляє (String);

– value – числове значення точки даних (Float).

Клас TableData містить інформацію для таблиць, що використовуються як інтерактивні елементи. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного запису даних таблиці (Int);

– title – назва або заголовок таблиці (String);

– headers – список заголовків колонок таблиці (список String);

– rows – список рядків у таблиці, де кожен рядок представлений як список значень (список списків String).

Клас SectionResponse містить рекомендовані відповіді для розділів, включаючи поради та інтерактивні елементи. Вона містить такі поля:

– id – унікальний ідентифікатор для кожної відповіді на розділ (Int);

– title – назва розділу, до якого стосується ця відповідь (String);

– tips – список рекомендованих порад для розділу (список String).

– recommendedInteractiveElements – список рекомендованих

інтерактивних елементів для розділу (список String).

Клас MainUiState представляє стан головного інтерфейсу користувача, відстежуючи поточний документ та налаштування інтерфейсу. Вона містить такі поля:

– id – унікальний ідентифікатор для кожного стану інтерфейсу користувача (Int);

– recentDocuments – список ідентифікаторів для нещодавно відкритих документів (Список String);

– currentDocument – посилання на поточний відкритий документ (String);

– showNewDocumentDialog – булеве значення, яке вказує, чи показувати діалог для створення нового документа (Boolean);

– showDocumentEditor – юулеве значення, яке вказує, чи відкрити редактор документа (Boolean);

– isLoading – булеве значення, яке вказує, чи завантажується інтерфейс (Boolean);

– errorMessage – повідомлення для відображення помилок у інтерфейсі (String).

2.8 Висновки

У другому розділі магістерської кваліфікаційної роботи було розроблено архітектуру системи, обрано патерн MVVM для її реалізації, описано сутність цього патерна. Розроблено та описано метод автоматичної генерації структури документа, метод рекомендацій наповнення документа, метод рекомендацій елементів візуалізації, метод роботи з елементами візуалізації у Android View інтерфейсі, розроблено модель системи та загальний алгоритм її роботи.

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ СИСТЕМИ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

Kotlin [13] – статично типізована мова програмування, розроблена компанією JetBrains. Вона була створена з метою покращення продуктивності розробників та надання сучасних мовних можливостей.

Kotlin повністю взаємодіє з Java, що означає, що код Java можна легко інтегрувати з кодом на Kotlin і навпаки. Завдяки цьому розробники можуть поступово переходити на Kotlin без необхідності переписувати весь код з нуля та використовувати уже існуючий код.

Kotlin працює на віртуальній машині Java (JVM), що дозволяє використовувати всі існуючі бібліотеки та фреймворки Java. Її синтаксис і мовні функції спрямовані на те, щоб допомогти розробникам писати більш безпечний та чистий код, зменшуючи кількість помилок та підвищуючи читабельність коду [14].

Kotlin має офіційну підтримку від Google для розробки Android, що робить його рекомендованою мовою для створення програм під цю платформу. З 2017 року Google оголосила Kotlin мовою "first-class" для Android, що означає, що нові API та інструменти розробляються з урахуванням Kotlin. Це значно спрощує процес розробки та дозволяє використовувати всі сучасні можливості мови.

Основні переваги Kotlin:

– зменшує кількість помилок, пов'язаних з null-посиланнями;

– код стає більш читабельним та менш об'ємним;

– підтримка Java легко інтегрується в існуючі проекти;

– наявність корутин спрощує роботу з асинхронними операціями;

– можливість розробки під різні платформи.

Архітектура Kotlin наведена на рисунку 3.1.

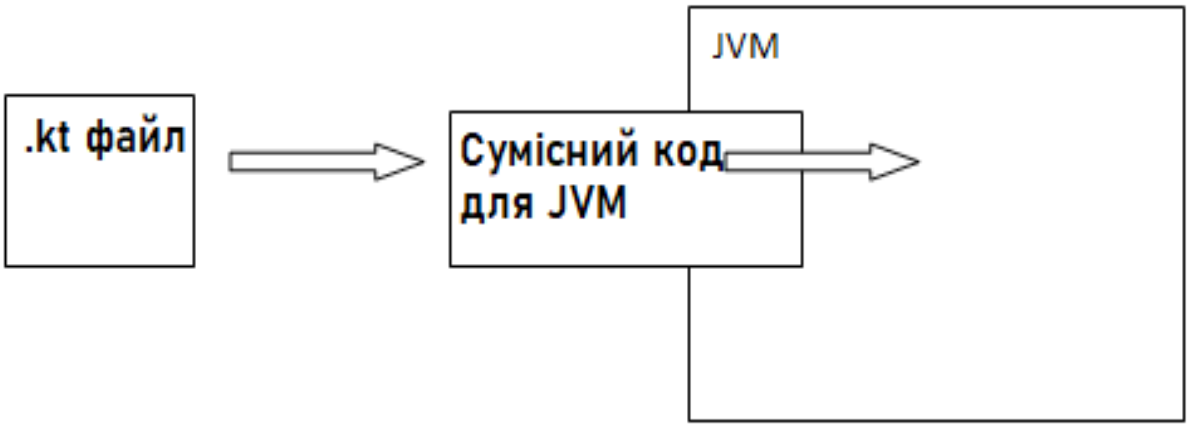

Рисунок 3.1 – Архітектура Kotlin

Мова охоплює парадигми як об’єктно-орієнтованого, так і функціонального програмування, пропонуючи такі функції, як функції вищого порядку, лямбда-вирази та корутини. Корутини в Kotlin спрощують асинхронне програмування, надаючи послідовний стиль кодування для асинхронних операцій, що покращує читабельність, зручність обслуговування та передбачуваність роботи асинхронного коду.

Dart [15] – це мова програмування загального призначення з відкритим кодом, розроблена компанією Google. Вона була створена для розробки фронтенд-застосунків і пропонує можливості для створення веб-додатків, серверних програм, десктопних застосунків та мобільних додатків. Dart оптимізована для створення інтерфейсів користувача з функціями, які підтримують "реактивне" програмування та декларативний стиль, що спрощує розробку сучасних користувацьких інтерфейсів.

Однією з ключових особливостей Dart є її використання у фреймворку Flutter, який забезпечує швидкі цикли розробки з такими функціями, як гаряче перезавантаження (hot reload). Ця можливість дозволяє розробникам бачити результати змін коду в режимі реального часу без перезапуску програми. Це значно прискорює процес розробки та підвищує продуктивність, особливо при роботі над великими застосунками, де компіляція може займати значний час.

Dart підтримує як інтерпретацію, так і компіляцію в нативний код, що

дозволяє досягти високої продуктивності на різних платформах. Мова також може бути транспільована в JavaScript, що робить її придатною для веб-розробки та дозволяє використовувати існуючі веб-технології та фреймворки. Це надає розробникам гнучкість у виборі платформи та технології для реалізації своїх проектів.

Основні переваги Dart:

– компіляція в нативний код забезпечує швидке виконання додатків.

– спрощує і прискорює процес розробки, дозволяючи миттєво бачити зміни.

– одна база коду для Android, iOS, веб та десктопних застосунків.

– легкий для вивчення, особливо для розробників, знайомих з мовами С-подібного синтаксису.

– зручні конструкції для роботи з асинхронними операціями (async/await) [16].

Архітектура Dart наведена на рисунку 3.2.

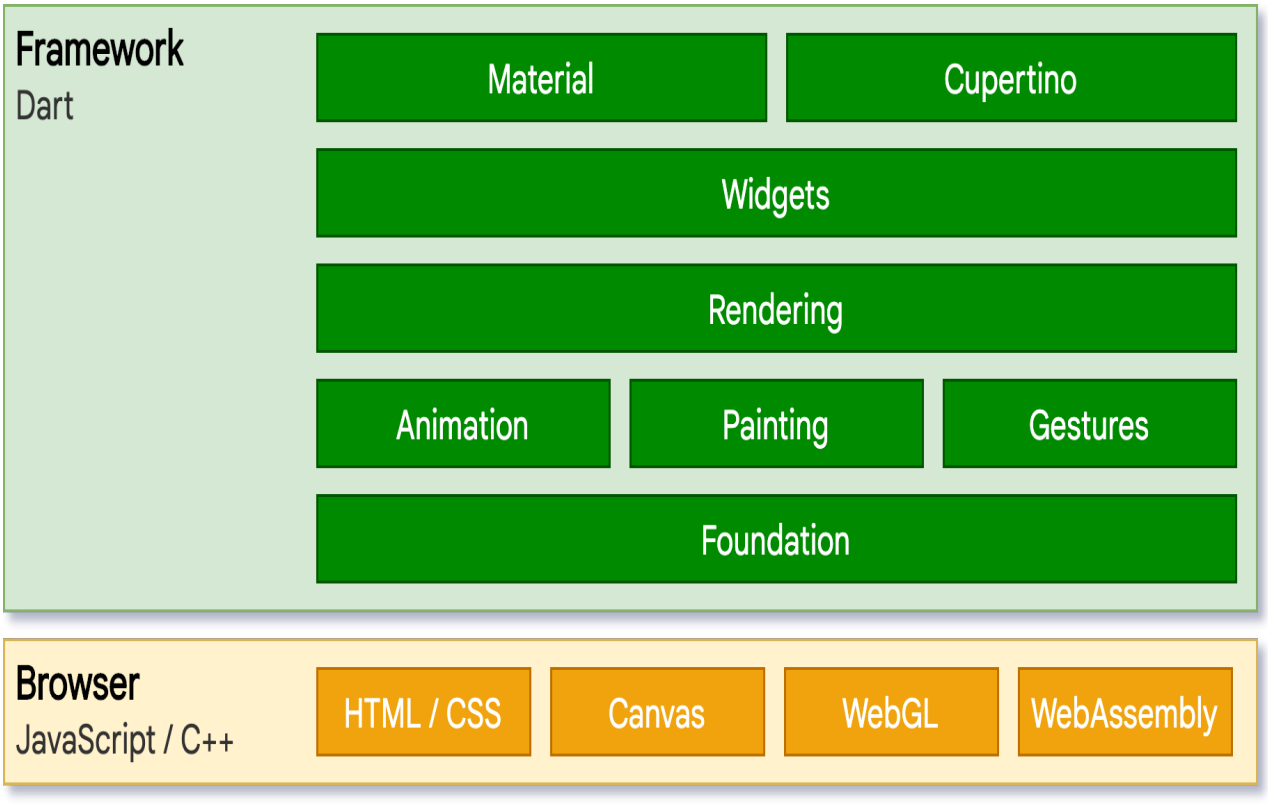

Рисунок 3.2 – Архітектура Dart

Dart підтримує як попередню, так і своєчасну (JIT) компіляцію. Попередня компіляція дозволяє скомпілювати код Dart у власний машинний код, що покращує час запуску та загальну продуктивність мобільних і настільних програм, а JIT-компіляція використовується під час розробки, щоб забезпечити функцію гарячого перезавантаження та швидкі цикли ітерації.

JavaScript [17] – це інтерпретована мова програмування, яка є однією з основних технологій Internet, поряд з HTML і CSS.

JavaScript відіграє ключову роль у контексті мультиплатформенної розробки завдяки своєму повсюдному поширенню у веб-браузерах та появі фреймворків, які розширюють його можливості.

Такі інструменти, як React Native, Node.js та Angular, дозволяють розробникам створювати власні мобільні програми для iOS і Android, а також серверні застосунки. Це робить JavaScript універсальним інструментом для розробки як клієнтської, так і серверної частин застосунків.

Фреймворк React Native надає можливості для створення повноцінних мобільних застосунків мовою JavaScript. Він дозволяє використовувати переваги цієї мови програмування, застосовуючи елементи веб-інтерфейсів у мобільних додатках.

React Native використовує нативні компоненти платформи, що забезпечує високу продуктивність та користувацький досвід, близький до нативних застосунків.

Node.js розширює можливості JavaScript на серверну сторону, дозволяючи створювати масштабовані та високопродуктивні серверні застосунки. Це середовище виконання на базі двигуна V8 від Google дає змогу розробникам використовувати одну і ту ж мову програмування як на клієнтській, так і на серверній стороні.

Angular, у свою чергу, є потужним фреймворком для створення веб-застосунків односторінкової архітектури (SPA), що спрощує розробку складних інтерфейсів користувача [18].

Архітектура JavaScript наведена на рисунку 3.3.

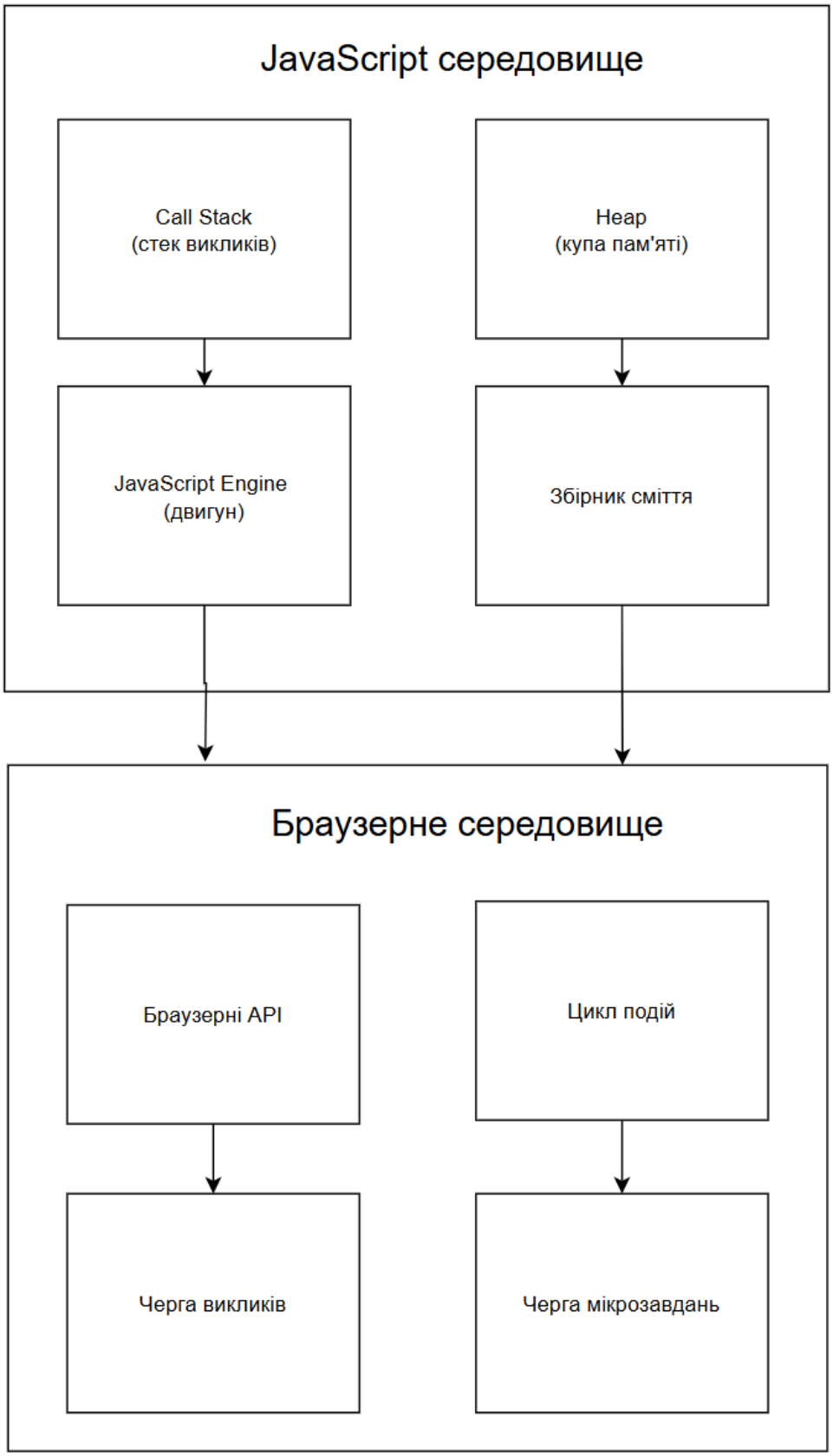

Рисунок 3.3 – Архітектура JavaScript

TypeScript [19], розроблений компанією Microsoft, додає до JavaScript

статичну типізацію та об'єктно-орієнтоване програмування. TypeScript компілюється у звичайний JavaScript, забезпечуючи сумісність із існуючими рішеннями на JavaScript. Він покращує якість коду та зручність обслуговування, особливо у великих програмах, де важливо мати чітку структуру та мінімізувати помилки типізації.

Основні переваги JavaScript:

– використовується як на клієнтській, так і на серверній стороні;

– багато бібліотек та фреймворків для різних задач;

– можливість розробки під веб, мобільні та десктопні платформи;

– постійне оновлення мови та інструментів;

– покращена якість коду завдяки статичній типізації у TypeScript.

C# [20] – це статично типізована об'єктно-орієнтована мова програмування, розроблена корпорацією Microsoft. Вона була створена як проста та сучасна мова загального призначення для створення широкого спектру програм, що працюють на платформі .NET. C# поєднує в собі найкращі риси C++ і Java, надаючи розробникам інструменти для розробки складних застосунків з високою продуктивністю.

C# охоплює кілька парадигм програмування, включаючи об'єктно-орієнтоване, функціональне та компонентно-орієнтоване програмування. Мова пропонує такі розширені функції, як генерики, Language Integrated Query (LINQ), зіставлення шаблонів і типи посилань із значенням nullable, які покращують виразність і безпеку коду. Керування пам'яттю здійснюється за допомогою збирача сміття (Garbage Collector), що зменшує ймовірність витоку пам'яті та пов'язаних з цим проблем.

Основні переваги C#:

– покращує безпеку та надійність коду;

– підтримка різних стилів програмування;

– розширені мовні функції: генерики, LINQ, асинхронне програмування;

– розробка для Windows, macOS, Linux, Android, iOS.

– регулярні оновлення від Microsoft та спільноти розробників [21].

Архітектура C# наведена на рисунку 3.4.

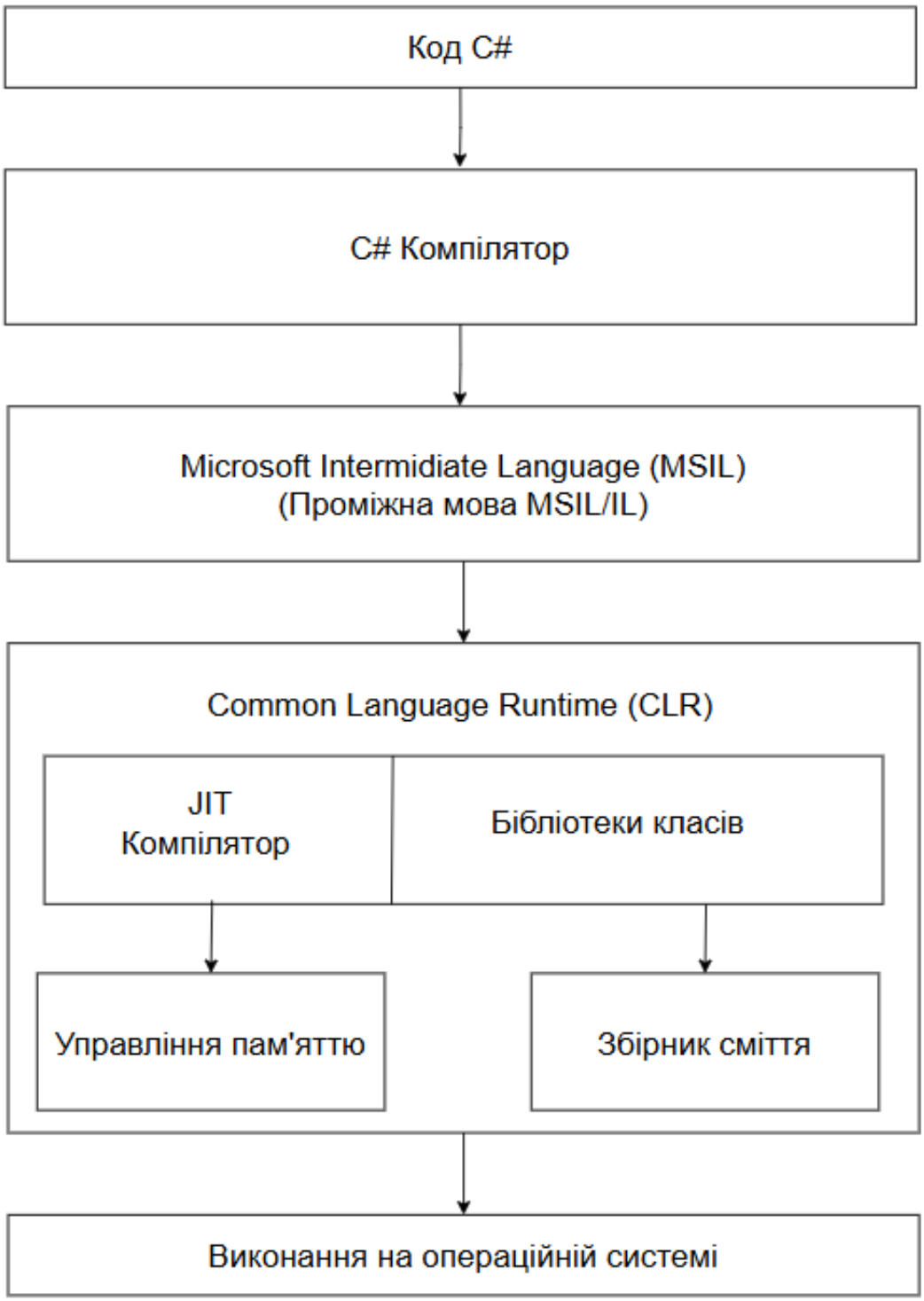

Рисунок 3.4 – Архітектура C#

Для порівняння характеристик описаних мов програмування, було сформовано таблицю 3.1 порівняння функціональних характеристик мов

програмування.

Таблиця 3.1 – Таблиця порівняння функціональних характеристик мов програмування

Характеристика	Kotlin	Dart	JavaScript	C#
Можливість розробляти кросплатформенні застосунки	1	1	1	1
Продуктивність при роботі з текстом	1	1	0	1
Бібліотеки для розробки текстових редакторів	1	1	1	1
Адаптація застосунків для роботи на мобільних пристроях	1	0	1	0
Генерація/редагування DOCX файлів	1	0	0	1
Загальна оцінка	5	3	3	4

Таким чином, згідно з наведеними вище даними, Kotlin має вищий сумарний коефіцієнт, ніж Kotlin, Dart та C#. Таким чином, Kotlin найкраще підходить для поставленого завдання.

Для аналізу середовищ розробки було обрано наступні: Android Studio, IntelliJ IDEA, AppCode.

IntelliJ IDEA [22], розроблена компанією JetBrains – розробниками мови Kotlin, – є інтегрованим середовищем розробки (IDE) для програмування на Java та Kotlin. Вона пропонує широкий набір функцій, які забезпечують надійну підтримку мультиплатформенних проектів Kotlin, дозволяючи розробникам легко писати, запускати та налагоджувати код на різних платформах, таких як JVM, Android, iOS та JavaScript.

Редактор коду IntelliJ IDEA оснащений розширеним автодоповненням коду, виявленням помилок у реальному часі та інструментами рефакторингу,

спеціально розробленими для Kotlin. Ці функції значно підвищують ефективність кодування та зменшують ймовірність помилок. Крім того, середовище надає можливості для статичного аналізу коду та автоматичного форматування, що допомагає підтримувати кодову базу в чистоті та відповідності до стандартів.

IntelliJ IDEA плавно інтегрується з такими інструментами збірки, як Gradle і Maven, підтримуючи Kotlin DSL для сценаріїв збірки. Це спрощує процес конфігурації проектів та управління залежностями. IDE також пропонує комплексні інструменти для модульного тестування та налагодження на різних платформах, що є необхідним для забезпечення надійності та продуктивності мультиплатформенних програм.

Основні переваги IntelliJ IDEA для розробки на Kotlin:

– інтелектуальні підказки та завершення коду;

– миттєве інформування про синтаксичні та логічні помилки;

– безпечне та швидке внесення змін у структуру коду;

– підтримка Gradle, Maven та Kotlin DSL;

– можливість налагодження коду на JVM, Android, iOS та JavaScript.

Інтерфейс IntelliJ IDEA наведено на рисунку 3.5.

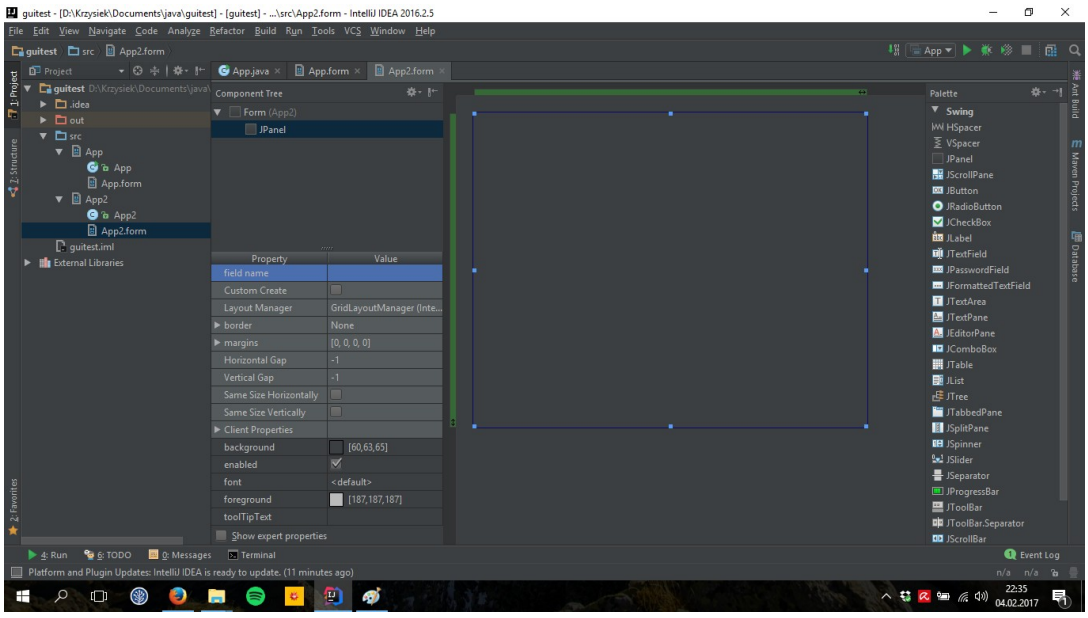

Рисунок 3.5 – Інтерфейс IntelliJ IDEA

Android Studio [23], офіційна інтегрована середовище розробки (IDE) від Google для розробки додатків під Android, побудована на основі IntelliJ IDEA та успадковує багато її потужних функцій. Вона додає спеціалізовані інструменти та вдосконалення, розроблені спеціально для розробки програм Android. Android Studio пропонує надійну підтримку проектів Kotlin Multiplatform Mobile (KMM), дозволяючи розробникам ефективно ділитися загальним кодом між програмами Android та iOS.

Android Studio містить набір специфічних для Android інструментів розробки, таких як розширений Редактор макета для проєктування інтерфейсів користувача, швидкий і багатофункціональний Емулятор Android для тестування програм на різних конфігураціях пристроїв, а також інструменти профілювання для ЦП, пам'яті та мережі, які є вирішальними для оптимізації продуктивності додатка. IDE тісно інтегрується з Gradle, підтримуючи Kotlin DSL для сценаріїв збірки, та надає розширені інструменти налагодження і тестування, включаючи підтримку специфічних для Android фреймворків тестування, таких як Espresso та UI Automator.

Такі функції, як Instant Run (нині відома як Apply Changes), дозволяють розробникам вносити зміни в код і ресурси в запущену програму без необхідності повного перезапуску, що значно прискорює процес розробки. Крім того, Android Studio пропонує бездоганну інтеграцію з такими службами Google, як Firebase, що полегшує додавання таких функцій, як аналітика, автентифікація та бази даних у реальному часі до додатків.

Android Studio доступна для Windows, macOS і Linux, надаючи комплексне середовище для розробників, які зосереджуються на мобільних додатках за допомогою Kotlin або Java. IDE підтримує різні інструменти продуктивності, такі як Lint для аналізу якості коду, та надає можливості для налаштування середовища відповідно до потреб розробника.

Основні переваги Android Studio:

– спеціалізовані інструменти для Android: емулятор, редактор макета, профайлер;

– ефективне спільне використання коду між Android та iOS.

– легке додавання Firebase та інших служб.

– підтримка Espresso, UI Automator.

– доступна на Windows, macOS та Linux.

AppCode [24], також розроблений компанією JetBrains, – це інтегроване середовище розробки (IDE), призначене в основному для розробки під iOS і macOS. Воно надає розробникам інструменти для написання якісного коду на Swift, Objective-C, C та C++, забезпечуючи глибоку інтеграцію з екосистемою Apple. AppCode пропонує підтримку проектів Kotlin Multiplatform, орієнтованих на платформи Apple, дозволяючи розробникам писати спільний код на Kotlin для додатків iOS разом із кодовими базами на Swift і Objective-C.

AppCode надає інтелектуальну допомогу в коді, включаючи розширене автодоповнення, аналіз коду в реальному часі та потужні інструменти рефакторингу. Глибока інтеграція з ланцюжком інструментів Xcode дозволяє розробникам легко керувати проектами та створювати конфігурації, використовуючи знайомі інструменти та процеси. IDE підтримує навігацію по коду, швидкий пошук та безліч інших функцій, що підвищують продуктивність розробки.

Хоча AppCode може не пропонувати таку широку підтримку Kotlin Multiplatform, як IntelliJ IDEA, особливо з точки зору інтеграції інструментів збірки, таких як Gradle з Kotlin DSL, воно все ж є цінним інструментом для розробників, які працюють в екосистемі Apple і бажають використовувати Kotlin для своїх програм iOS. IDE підтримує специфічні для iOS функції, такі як Storyboard та XIB редактори, а також надає інструменти для роботи з Core Data та іншими технологіями Apple.

AppCode містить такі функції, як автодоповнення коду, підсвічування синтаксису та інструменти налагодження, специфічні для розробки під iOS. Він також підтримує фреймворки модульного тестування, які використовуються в середовищі розробки Apple, такі як XCTest та Quick/Nimble, що дозволяє розробникам забезпечувати якість свого коду. Інструменти профілювання

допомагають оптимізувати продуктивність додатків, аналізуючи використання пам'яті та ЦП.

Основні переваги AppCode:

— повна сумісність з Xcode, підтримка Swift, Objective-C, C та C++;

— розширене автодоповнення, аналіз коду, рефакторинг;

— можливість використовувати Kotlin для розробки під iOS;

— підтримка XCTest, інтегрований налагоджувач;

— робота з Storyboard, XIB, Core Data.

Для порівняння можливостей цих середовищ розробки, було сформовано таблицю 3.2.

Таблиця 3.2 – Таблиця порівняння можливостей середовищ розробки для Kotlin

Характеристика	IntelliJ IDEA	Android Studio	AppCode
Інструменти розробки для мобільних пристроїв	0	1	1
Підтримка мови Kotlin	1	1	1
Ефективне використання ресурсів	1	1	1
Функціонал для рефакторингу коду	1	1	1
Вбудована підтримка систем контролю версій	1	1	1
Інтегровані фреймворки тестування	1	1	0
Загальна оцінка	5	6	5

Таким чином, Android Studio отримує найвищу загальну оцінку серед порівнюваних IDE. Це означає, що це середовище розробки є найкращим для розробки системи.

3.2 Розробка головного модуля системи

Головний модуль є вхідною точкою у систему, з якої починається взаємодія користувача. Цей модуль дозволяє користувачеві переглядати список

останніх файлів, з якими він взаємодіяв, та здійснювати пошук необхідних документів. Список файлів містить інформацію про назву файлу, його тему та призначення для кращого розуміння змісту кожного з документів без необхідності відкривати кожен із них.

З цього модуля можна створити новий документ при натисканні на відповідну кнопку, після чого користувача буде переадресовано на модуль редагування й формування текстових документів.

Для розширення функціоналу модуля, розроблено функціонал для бокового меню, яке надає можливість перейти з головного модуля на вікно перегляду видалених файлів, а також обрати конкретну папку для перегляду, якщо у неї додані файли. Окрім цього, користувач може додавати та вилучати існуючі папки. Поділ на папки розроблений для сортування документів за різним призначенням для того, аби надати функціонал для швидшого пошуку необхідного документа.

Алгоритм роботи головного модуля системи наведено на рисунку 3.6.

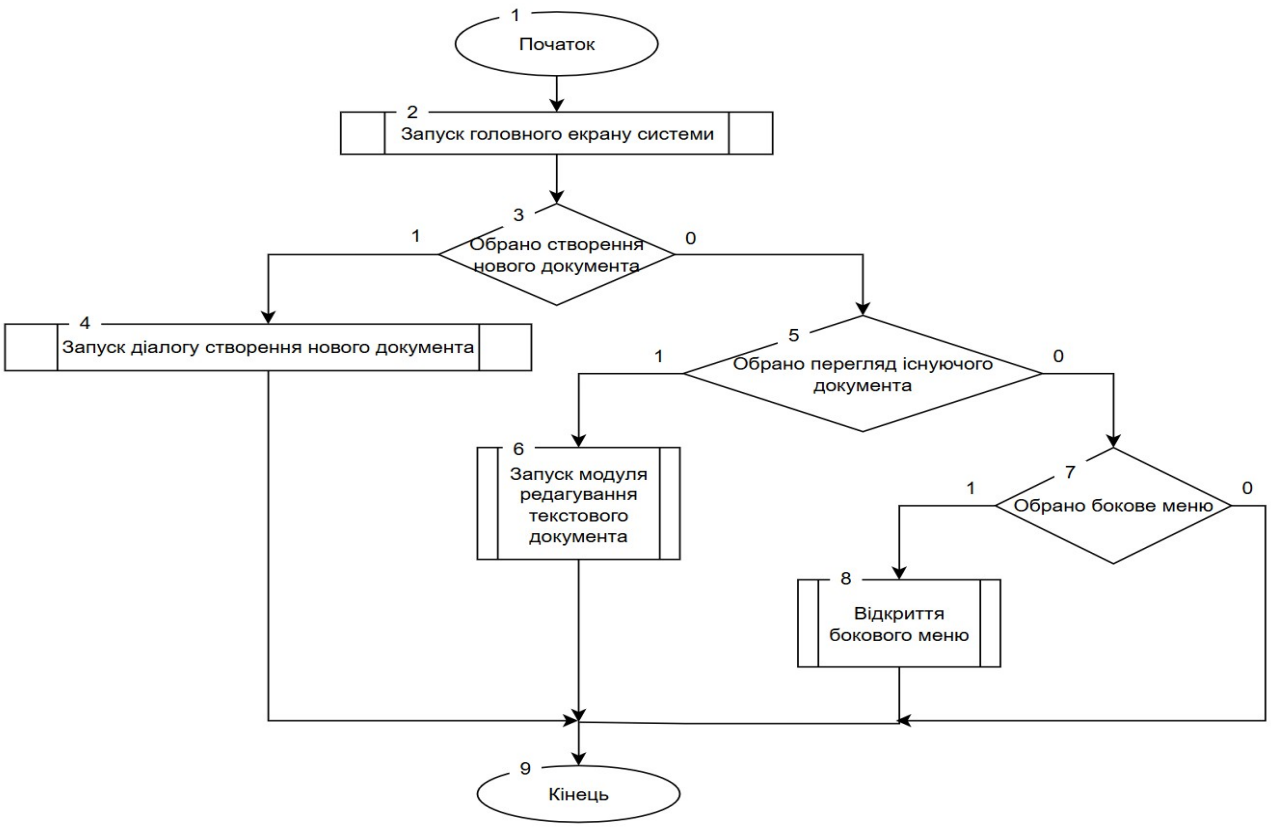

Рисунок 3.6 – Загальний алгоритм роботи головного модуля системи

3.3 Розробка модуля редагування й формування текстових документів

Модуль редагування й формування текстових документів є тим модулем, з яким користувач найбільше взаємодіє. Він дозволяє користувачеві створювати нові документи, вказувати їхнє призначення й тему, на основі чого модуль буде формувати структуру документа, яка включає в себе розділи документа і рекомендації для наповнення кожного розділу певним вмістом. Користувач може створювати нові документи, редагувати уже існуючі, а також завантажити свої документи для подальшої роботи з ними. Крім цього, реалізовано можливість роботи із зображеннями, таблицями та графіками.

Для роботи модуля, розроблено компонент EditorActivity. Цей компонент забезпечує інтерфейс, з яким взаємодіє користувач для виконання взаємодії із модулем редагування й формування текстових документів.

Окрім EditorAcvitity, розроблено та імплементовано інші компоненти, які наведені на діаграмі класів (рисунок 3.7).

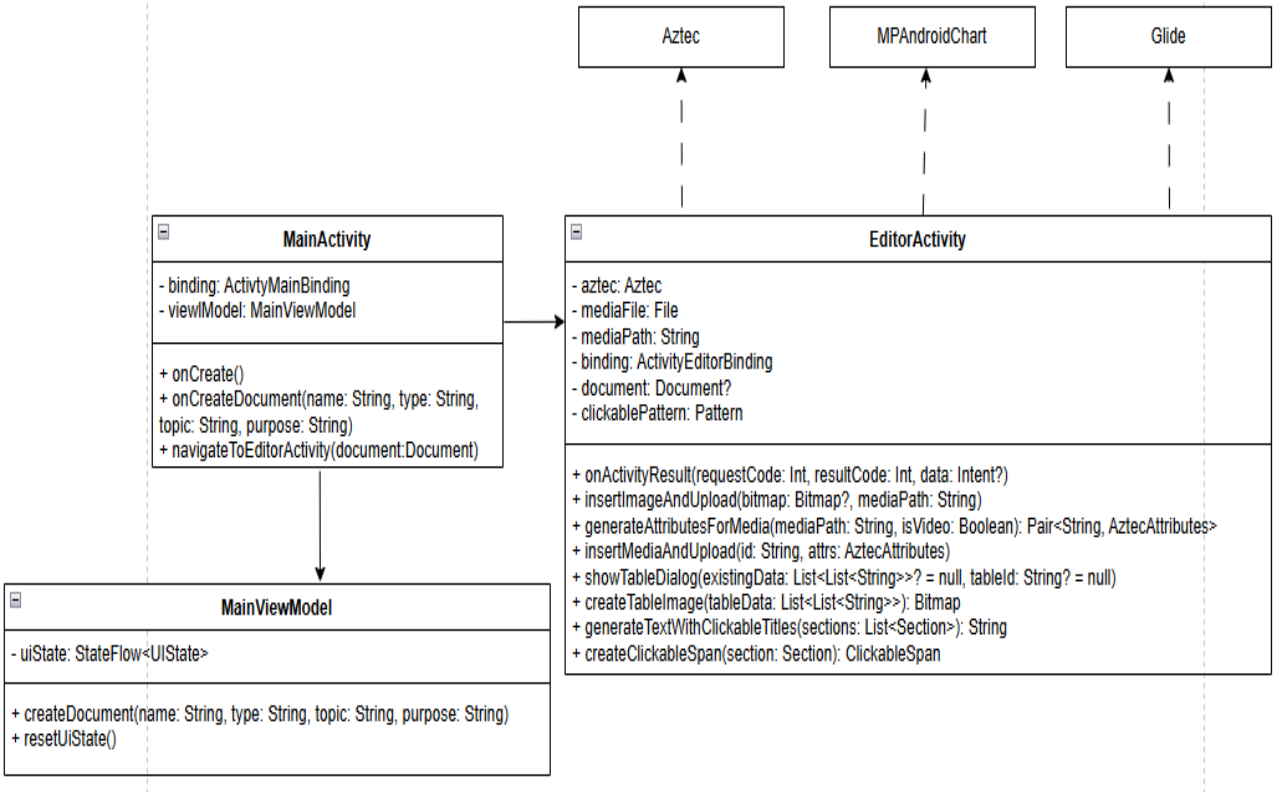

Рисунок 3.7 – Діаграма класів системи

EditorActivity являє собою клас, який включає такий набір параметрів:

– aztec: Aztec – об’єкт класу Aztec бібліотеки Aztec, який надає інтерфейс для розробки текстового редактора та функціонал для редагування текстових документів;

– mediaFile: File – допоміжний об’єкт, який використовується для роботи з файлами у модулі редагування й формування текстових документів, таких як рисунки, таблиці чи графіки;

– mediaPath: String – допоміжний об’єкт, який використовується для роботи із посиланнями на файли, що додаються до документу;

– binding: ActivityEditorBinding – об’єкт, який надає функціонал для взаємодії з елементами інтерфейсу;

– document: Document? – об’єкт, який містить в собі опис текстового документа.

Окрім цього, цей клас містить такі методи:

– insertImageAndUpload(bitmap: Bitmap?, mediaPath: String) – метод, що застосовується для вставлення елементів візуалізації в документ;

– showTableDialog(existingData: List<List<String>>? = null, tableId: String? = null) – метод, що застосовується для виведення діалогу для роботи із таблицею;

– createTableImage(tableData: List<List<String>>): Bitmap – метод для перетворення таблиці у зображення;

– generateTextWithClickableTitles(sections: List<Section>): String – метод для створення інтерактивного тексту, при натисканні на яких з’являються поради щодо наповнення текстового документу;

– createClickableSpan(section: Section): ClickableSpan – допоміжна функція, яка застосовується при створенні інтерактивного тексту.

Також розроблено клас MainActivity, який являє собою головний модуль системи. Він містить такі параметри:

– binding: ActivityMainBinding – об’єкт, який надає функціонал для взаємодії з елементами інтерфейсу;

– viewModel: MainViewModel – об’єкт класу MainViewModel, який містить в собі бізнес-логіку роботи системи.

Окрім цього, клас включає такі методи:

– onCreateDocument(name: String, type: String, topic: String, purpose: String) – метод, який активується при створенні документа;

– navigateToEditorActivity (document: Document) – виконується після створення документа та перенаправляє користувача у EditorActivity.

Для роботи текстового редактора використовуються бібліотеки Aztec, MPAndroidChart, Glide.

Aztec – бібліотека, розроблена компанією Wordpress для розробки текстових редакторів на операційній системі Android.

Вона надає інтерфейс для редагування тексту, базовий функціонал для стилізації тексту, можливість додавати зображення у поле для редагування тексту. Користувачі можуть конвертувати HTML-код у стилізований текст в основному редакторі з використанням окремої версії текстового редактора. Також, ця бібліотека дозволяє користувачам адаптувати наданий інтерфейс під свої потреби.

MPAndroidChart – бібліотека, що розроблена створення різноманітних графіків та діаграм у застосунках на ОС Android. Вона підтримує інтерактивні функції, такі як масштабування, прокрутка та анімація, що дозволяє будувати лінійні графіки, стовпчасті діаграми, комбіновані графіки, кругові діаграми, та надає широкий функціонал для їх налаштування.

Glide – бібліотека для парсингу та завантаження зображень. Найбільше вона використовується для завантаження зображень з інтернету за посиланням, з їх подальшим розміщенням у спеціально визначене місце в застосунку, але може і застосовуватися для завантаження зображень із внутрішньої пам’яті пристрою, якщо надати внутрішнє посилання на місце розміщення зображення у пам’яті.

Для наочної демонстрації процесу завантаження зображення у текстовий документ, було побудовано таблицю 3.3 послідовності виконання дій.

Таблиця 3.3 – Послідовність виконання дій при завантаженні зображення у текстовий редактор

Крок	Відправник	Отримувач	Дія
1	Користувач	EditorActivity	Користувач натискає кнопку для вставлення зображення.
2	EditorActivity	PermissionUtils	Перевірка наявності необхідних дозволів для завантаження зображень.
3	PermissionUtils	EditorActivity	Повернення результату щодо наявності дозволів.
4	EditorActivity	Користувач	Запит у користувача необхідних дозволів.
5	Користувач	EditorActivity	Користувач надає необхідні дозволи.
6	EditorActivity	Галерея	Запуск галереї для вибору зображення
7	Галерея	EditorActivity	Повернення обраного зображення.
8	EditorActivity	BitmapFactory	Декодування зображення у об’єкт Bitmap.
9	BitmapFactory	Aztec Editor	Виклик `insertImageAndUpload` для вставлення зображення.
10	AztecEditor	Користувач	Вставлене зображення відображається користувачеві.

Діаграма послідовності описаного процесу наведена на рисунку 3.8.

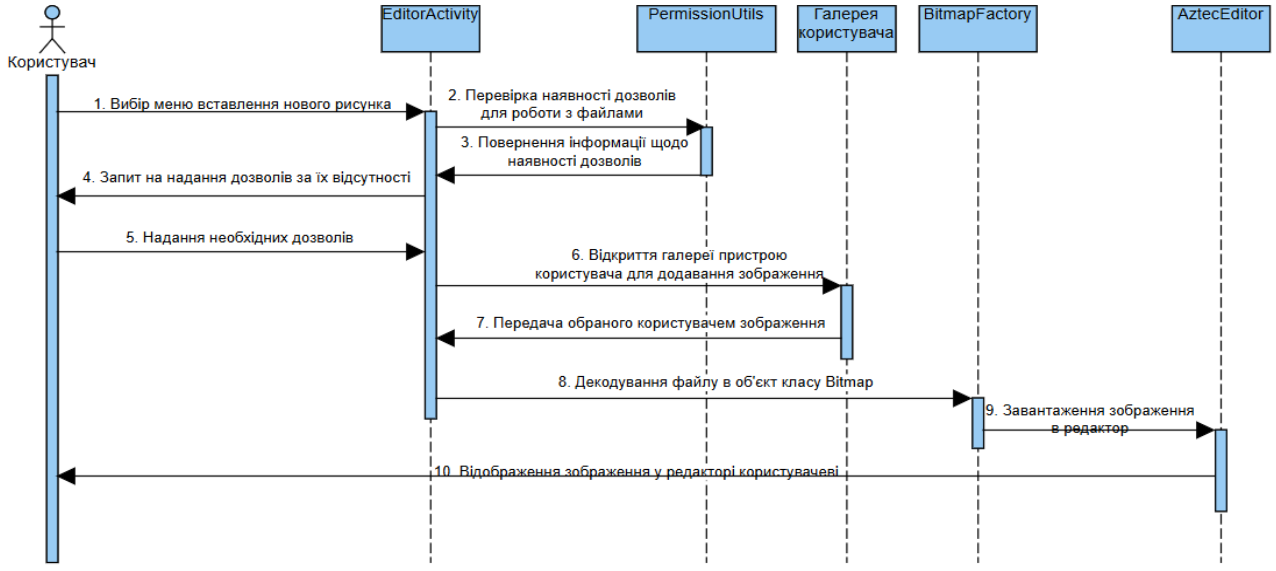

Рисунок 3.8 – Діаграма послідовності процесу вставлення зображення у документ

3.4 Розробка інтерфейсу системи

Для застосунків ОС Android існує два основних підходи до розробки інтерфейсу користувача: Android View та Jetpack Compose.

Android View [25] – традиційний підхід до розробки інтерфейсів Android застосунків. В ньому екрани користувацького інтерфейсу описуються в окремих файлах за допомогою XML нотації, після чого кожен елемент інтерфейсу, з яким відбуватиметься взаємодія по ходу роботи програми, потрібно ініціалізувати, створивши для нього окрему змінну та зв’язати із елементом XML файла.

Перевагами Android View є:

– більш детальна документація, яка дозволяє краще зрозуміти та вивчити специфіку роботи різних елементів інтерфейсу;

– більший набір елементів інтерфейсу для реалізації різноманітних застосунків;

– підтримка старіших версій ОС Android.

При цьому, Android View має і недоліки:

– вимагає більше коду для реалізації та підтримки необхідного функціоналу;

– погана масштабованість великих складних застосунків, адже нагромадження XML коду ускладнює його подальшу розробку та збільшує ймовірність помилок при розробці.

Jetpack Compose [26] – сучасний фреймворк для створення інтерфейсу користувача в Android. На відміну від Android View, він не потребує XML коду, натомість же розробка екранів інтерфейсу відбувається за допомогою функцій мовою Kotlin.

Переваги Jetpack Compose:

– менша кількість коду, порівняно із Android View;

– декларативний підхід до розробки інтерфейсу, що означає створення інтерфейсу у вигляді окремих функцій. Це спрощує читання та розуміння коду, дозволяє тестувати інтерфейс різних екранів окремо один від одного за

допомогою спеціальних Preview (попередній перегляд) функцій;

– модульність та повторне використання компонентів.

Попри це, Jetpack Compose також має недоліки:

– можлива несумісність зі старішими фреймворками та бібліотеками;

– новизна фреймворку, через що деякі необхідні функції, які присутні на Android View, можуть бути відсутні.

Jetpack Compose загалом є сучаснішим підходом для розробки Android-інтерфейсу, так як постійно оновлюється, надаючи можливість розробляти привабливі та сучасні інтерфейси, легше інтегрувати логіку роботи застосунків із інтерфейсом.

При цьому, Jetpack Compose не володіє достатнім функціоналом для розробки системи для редагування й формування текстових документів. Так, наприклад, Jetpack Compose не дозволяє додавати та відображати зображення у полі для редагування тексту, що суттєво обмежує можливості редагування документів.

Також відсутня можливість створення інтерактивного тексту, при натисканні на який відбувався б перехід по посиланням чи відкриття діалогових вікон, що не дозволяє розробку підказок при написанні тексту для певної частини документа.

Таким чином, для розробки системи для редагування й формування текстових документів було обрано Android View, так як він підтримує описаний функціонал, наявність якого є важлива для роботи з текстовими документами.

Було розроблено інтерфейс головного вікна системи, схему якого наведено на рисунку 3.9.

Складові головного вікна системи:

1. Кнопка для відкриття бокового меню.
2. Назва системи.
3. Рядок пошуку.
4. Список документів.
5. Кнопка для створення нового документа.

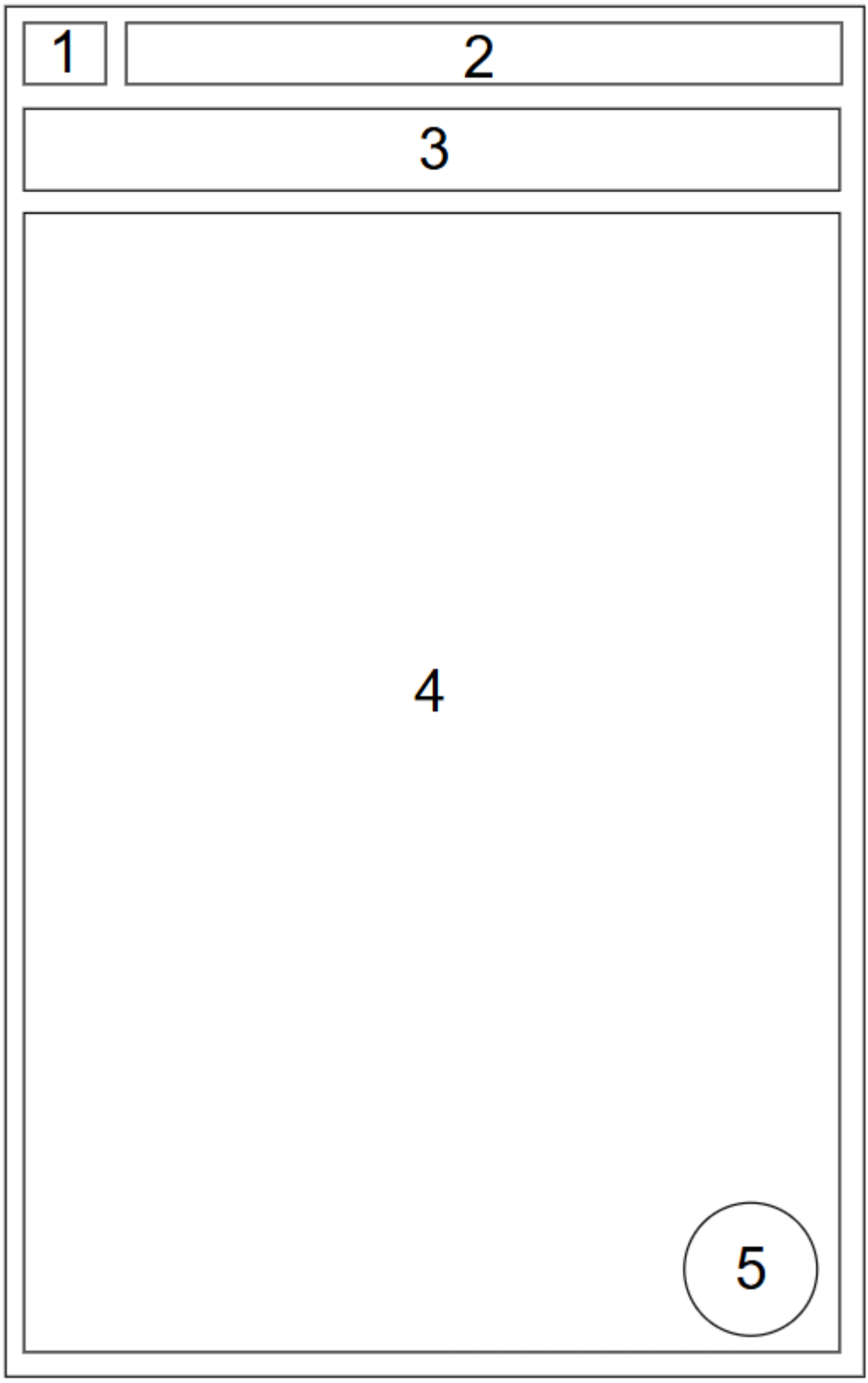

Рисунок 3.9 – Структурна схема головного вікна системи

Було розроблено структурну схему вікна для редагування текстових документів, яка наведена на рисунку 3.10.

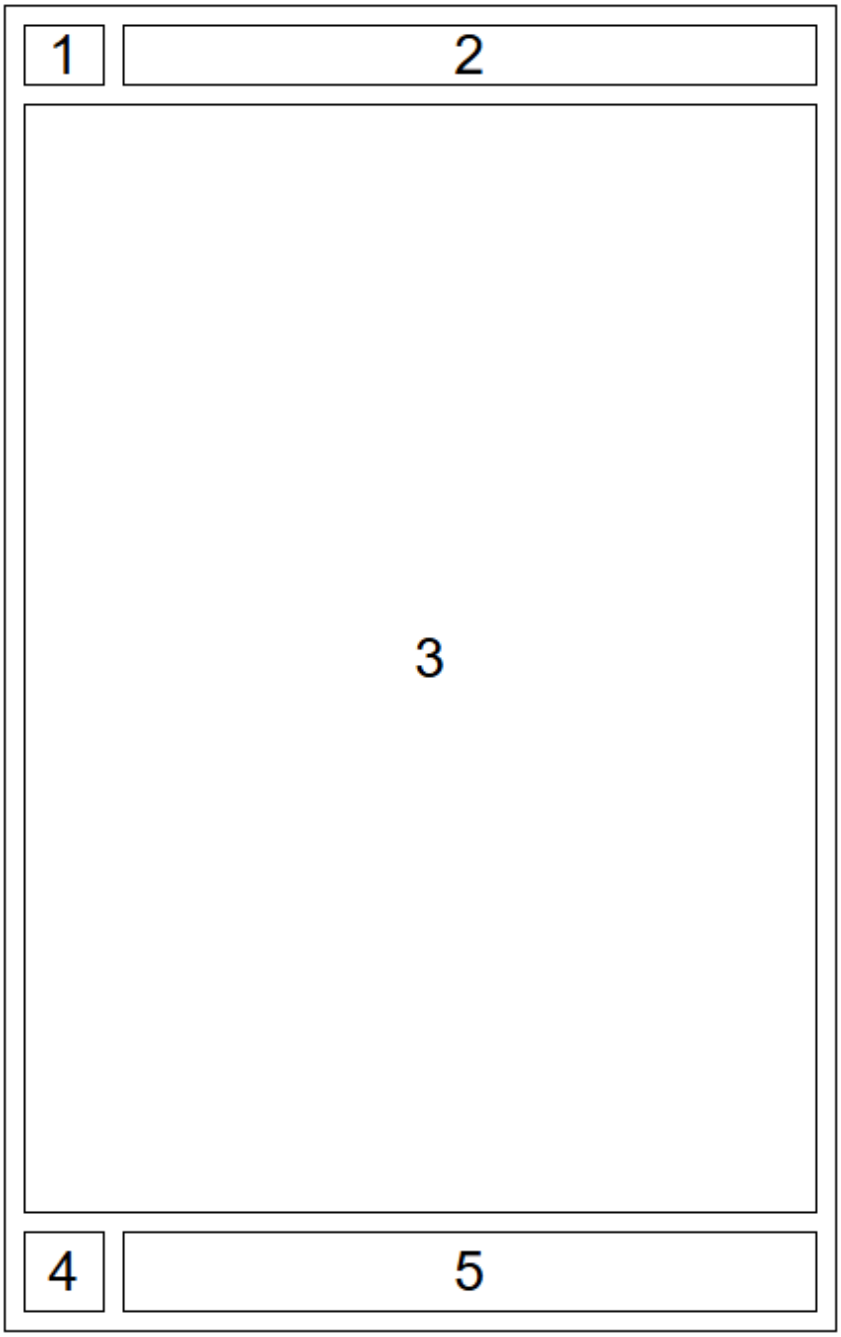

Рисунок 3.10 – Структурна схема вікна редагування текстових документів

Елементи вікна:

1. Кнопка для переходу на попереднє вікно.
2. Поле для назви файлу.
3. Поле редагування документа.
4. Кнопка для додавання елементів візуалізації.
5. Панель для форматування тексту.

Було розроблено структурну схему інтерфейсу бокового меню головного

вікна системи, яка наведена на рисунку 3.10.

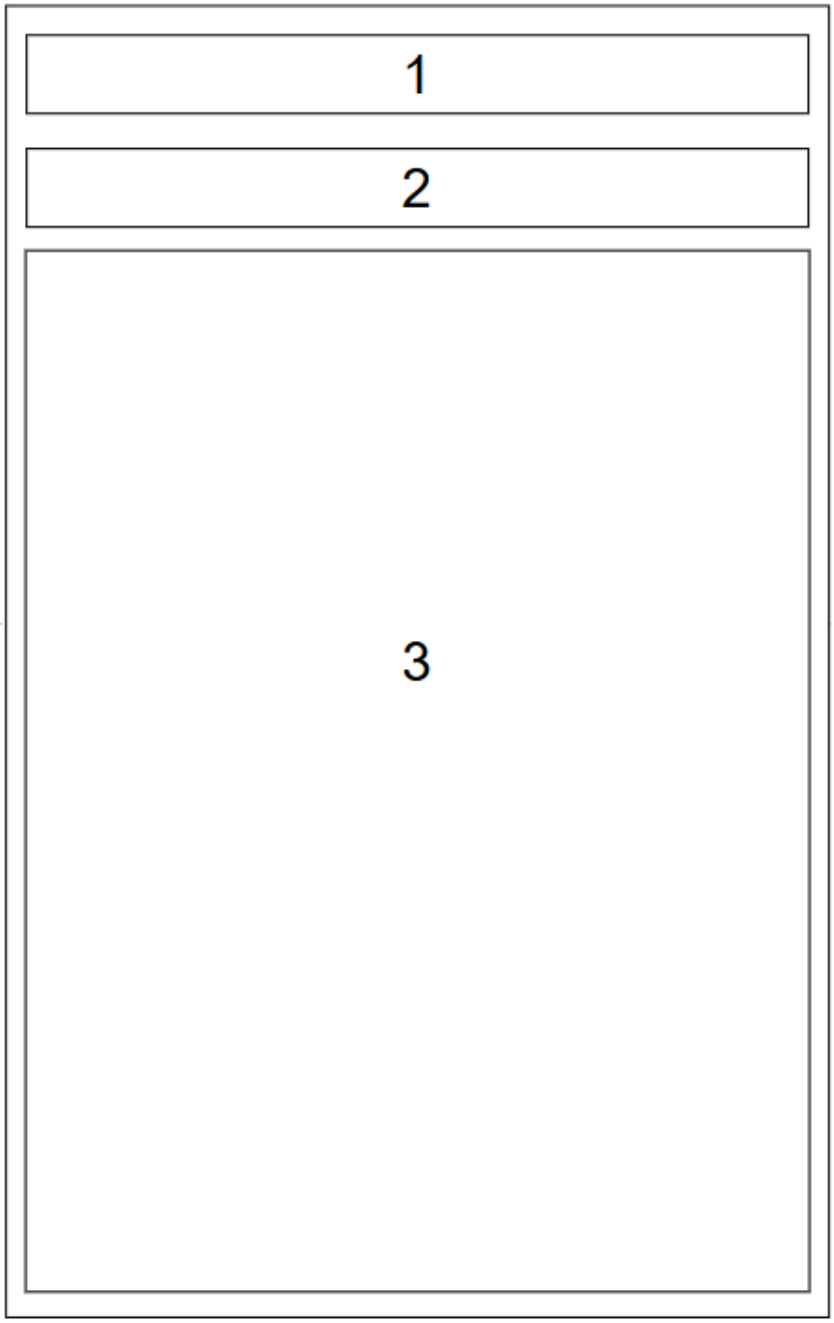

Рисунок 3.10 – Структурна схема бокового меню головного вікна системи

Складові бокового меню:

1. Кнопка для вибору усіх файлів.
2. Кнопка для вибору видалених файлів.
3. Меню для папок.

Також було розроблено панель для форматування тексту (рисунок 3.11 та

рисунок 3.12).

Рисунок 3.11 – Панель для форматування тексту (частина 1)

Рисунок 3.12 – Панель для форматування тексту (частина 2)

Було розроблено інтерфейс діалогового вікна для створення таблиці. Воно містить в собі поля для введення кількості рядків та стовпців таблиці, кнопки для закриття діалогового вікна та відміни створення таблиці, та кнопки підтвердження та додавання таблиці до документа. Дизайн діалогового вікна наведено на рисунку 3.13.

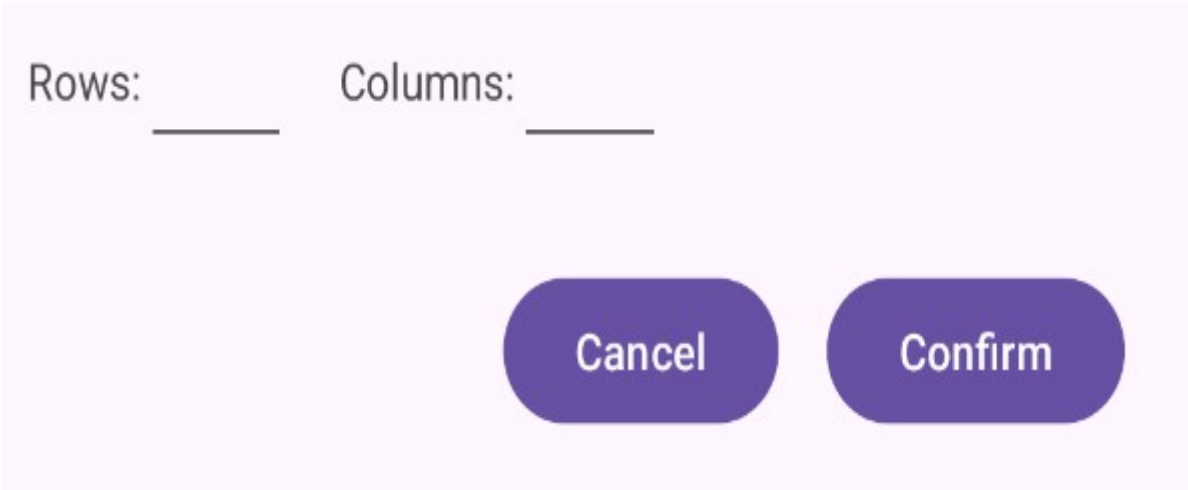

Рисунок 3.13 – Дизайн діалогового вікна для додавання нової таблиці

Розроблено дизайн діалогового вікна для створення кругової діаграми. Воно містить 2 поля, в першому з яких потрібно вводити значення через кому, а у другому – назву цих значень через кому. Дизайн наведено на рисунку 3.14.

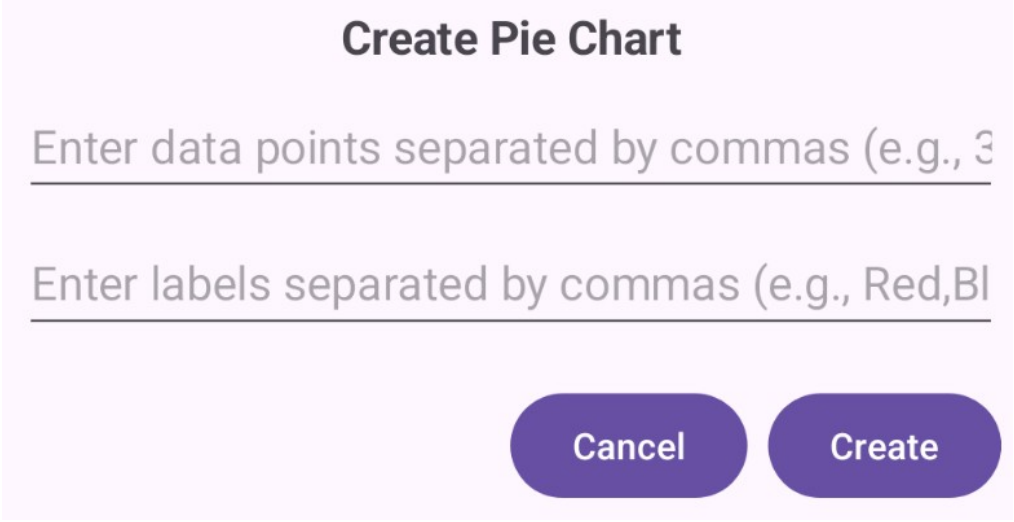

Рисунок 3.14 – Дизайн діалогового вікна створення кругової діаграми

Розроблено інтерфейс діалогового вікна створення нового документу. Воно містить 4 текстових поля для назви файлу, типу документа, теми та призначення.

Інтерфейс діалогового вікна створення нового документу наведено на рисунку 3.15.

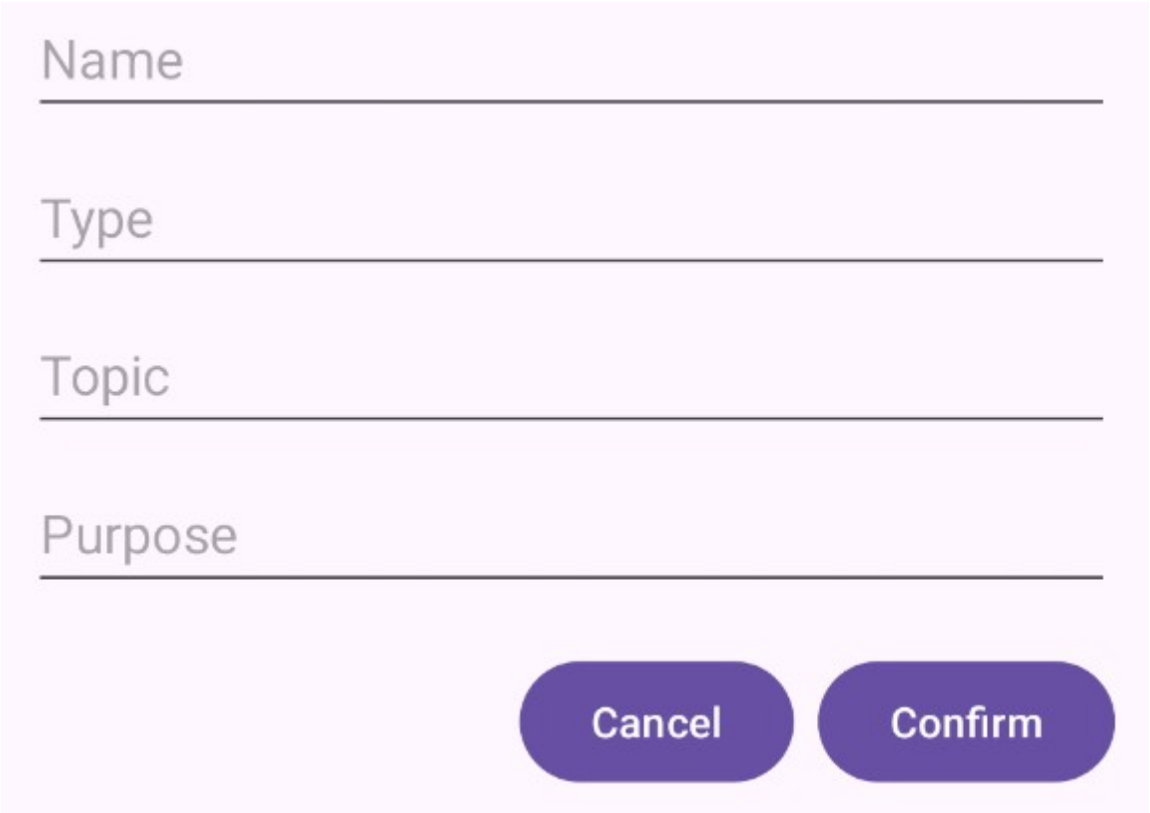

Рисунок 3.15 – Інтерфейс діалового вікна створення нового документу

3.5 Висновки

У третьому розділі магістерської кваліфікаційної роботи було проведено порівняльний аналіз мов програмування та середовищ розробки, було обрано мову програмування Kotlin та середовище розробки Android Studio для створення системи. Описано розроблений головний модуль та модуль редагування текстових документів, продемонстровано порядок виконання процесів при взаємодії користувача з ними. Проведено порівняння двох підходів до розробки інтерфейсу Android застосунків – Android View та Jetpack Compose. В результаті, було прийнято рішення розробляти інтерфейс з використанням Android View, який більш пристосований до реалізації функціоналу системи для редагування й формування текстових документів. Було розроблено структурні схеми та дизайн інтерфейсу користувача.

4 ТЕСТУВАННЯ СИСТЕМИ

4.1 Аналіз методів тестування програмного забезпечення

Тестування мобільних застосунків є невід'ємною частиною сучасного процесу розробки, оскільки якість продукту безпосередньо впливає на досвід користувача та його задоволення. У світі, де мобільні пристрої стали повсякденним інструментом для комунікації, розваг та бізнесу, важливо забезпечити надійність і функціональність застосунків [27].

До основних причин для проведення тестування застосунків можна віднести:

1. виявлення та виправлення помилок;
2. забезпечення високої якості користувацького досвіду;
3. перевірка відповідності вимогам та специфікаціям;
4. підвищення безпеки;
5. оптимізація продуктивності;
6. забезпечення сумісності та крос-платформенності;
7. зменшення витрат на підтримку та обслуговування [28].

Існує велика кількість методів тестування, кожен з яких має свої особливості, переваги та недоліки.

Ручне тестування передбачає, що тестувальник особисто перевіряє функціональність застосунку, виконуючи різні сценарії використання. Це дозволяє реагувати на несподівані помилки та краще оцінити користувацький досвід, оскільки тестувальник безпосередньо взаємодіє з інтерфейсом. Перевагами є відсутність потреби у складних інструментах чи навичках програмування, що робить його доступним [29].

Автоматизоване тестування використовує спеціально розробленів інструменти та скрипти для автоматичного виконання тестів, що забезпечує швидкість і ефективність при повторному виконанні тестів, зменшуючи людський фактор у процесі тестування. До недоліків можна віднести високі початкові витрати на налаштування, потреба в навичках програмування та менш

ефективне виявлення проблем з інтерфейсом користувача.

Функціональне тестування спрямоване на перевірку відповідності застосунку вимогам та специфікаціям. Цей метод тестування гарантує, що всі функції працюють коректно, і допомагає виявити критичні помилки на ранніх стадіях розробки. Однак, цей метод не враховує аспектів продуктивності чи безпеки і може бути трудомістким без автоматизації процесу.

Тестування методом «чорної скриньки» (Black Box Testing) є одним з основних підходів до тестування програмного забезпечення, при якому тестувальник оцінює функціональність системи без знання її внутрішньої структури або коду. Тестувальник взаємодіє з системою, базуючись на специфікаціях та вимогах, перевіряючи, чи відповідає поведінка програми очікуваним результатам.

До особливостей цього методу можна віднести фокус на функціональності, так як тестувальник перевіряє, чи система виконує свої функції згідно з вимогами, відсутність знань про код, що дозволяє зосередитися на кінцевому результаті.

До переваг методу «чорної скриньки» можна віднести ефективність у виявленні функціональних помилок, та симуляцію поведінки користувача, що дозволяє виявити проблеми, які можуть виникнути при реальному використанні системи [30].

Таким чином, враховуючи потреби в тестуванні системи, та особливості описаних методів, було прийнято рішення провести тестування розробленої системи з використанням методу чорної скриньки.

4.2 Тестування розробленої системи

Тестування системи розпочинається із її головного екрана, який відкривається при початковому запуску системи. Вона являє собою список файлів, що були створені з її використанням. Головний екран системи наведено на рисунку 4.1.

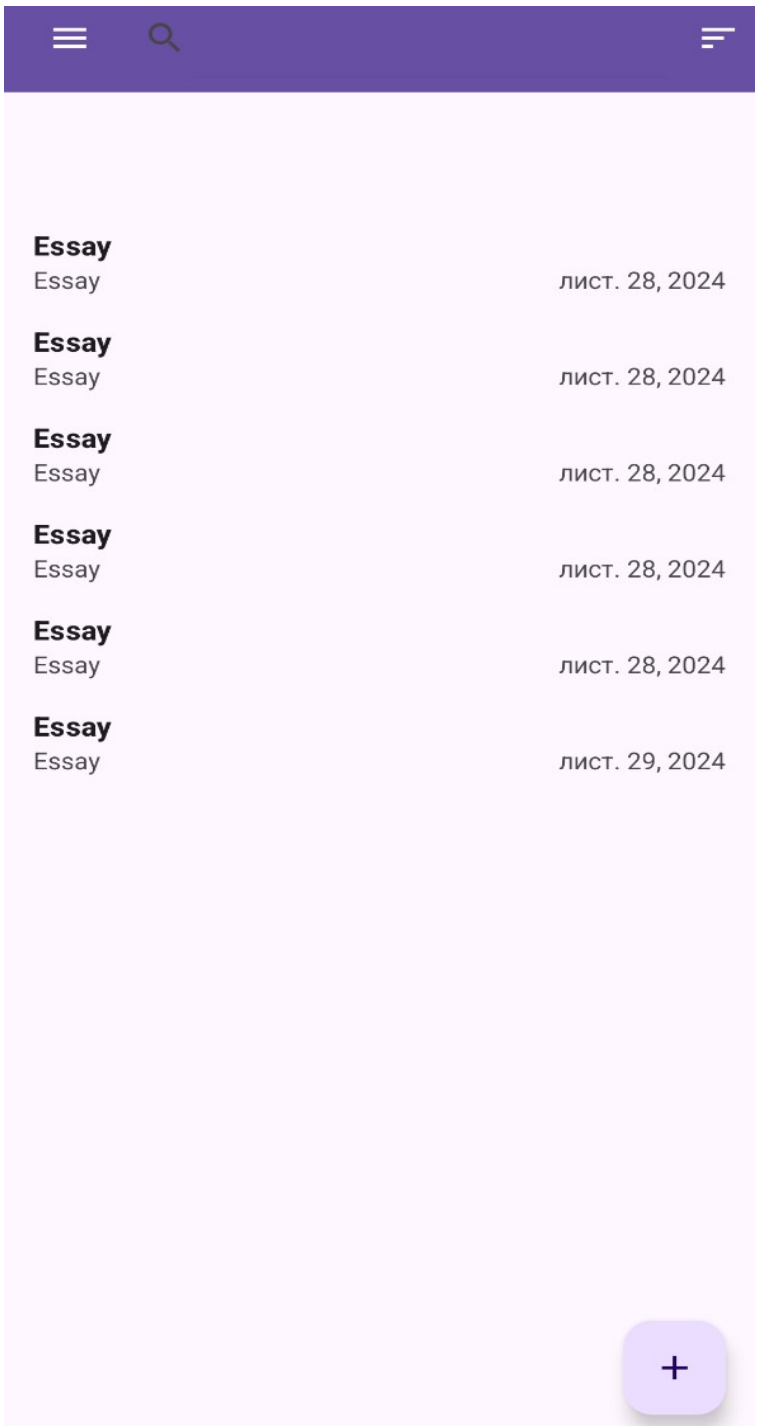

Рисунок 4.1 – Головний екран системи

З метою тестування, спершу було згенеровано кілька тестових файлів. Їх можна видалити, провівши обраний файл свайпом вліво, що перемістить їх у розділ з вилученими файлами. При цьому внизу екрана з’явиться відповідне повідомлення, яке наведене на рисунку 4.2. Протягом деякого часу, видалений файл можна повернути. Якщо ж цього не буде зроблено, повідомлення автоматично зникне.

Рисунок 4.2 – Повідомлення про видалення файлу

Оновлений список файлів після вилучення деяких файлів продемонстровано на рисунку 4.3.

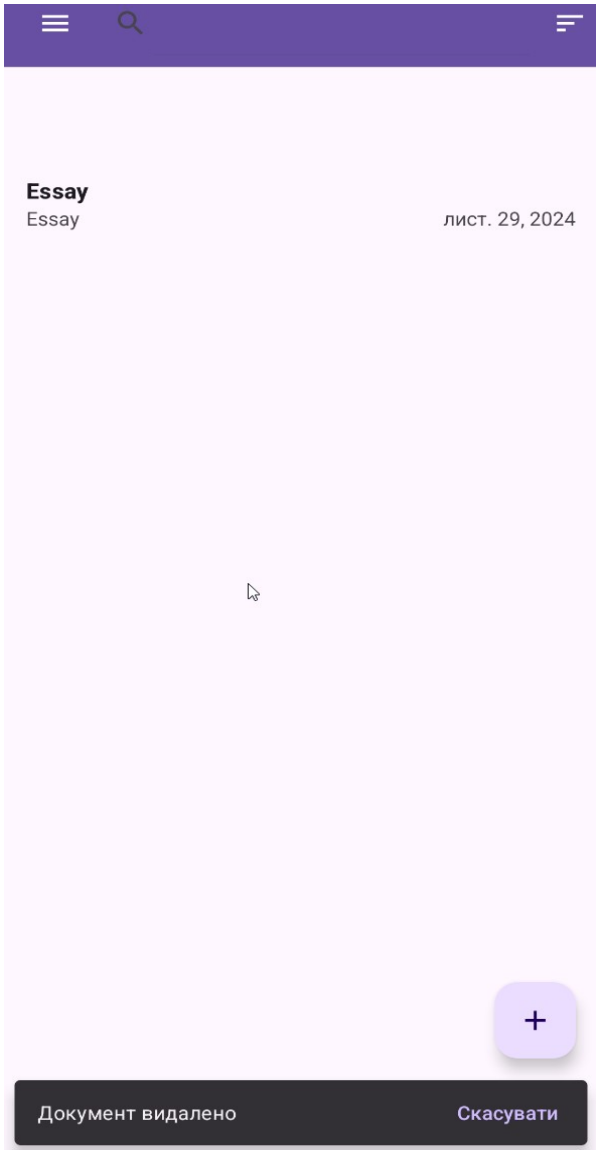

Рисунок 4.3 – Оновлений список файлів на головному екрані

На рисунку 4.4 продемонстровано екран з видаленими файлами. Як можна переконатися, усі вилучені файли перемістилися у цей розділ.

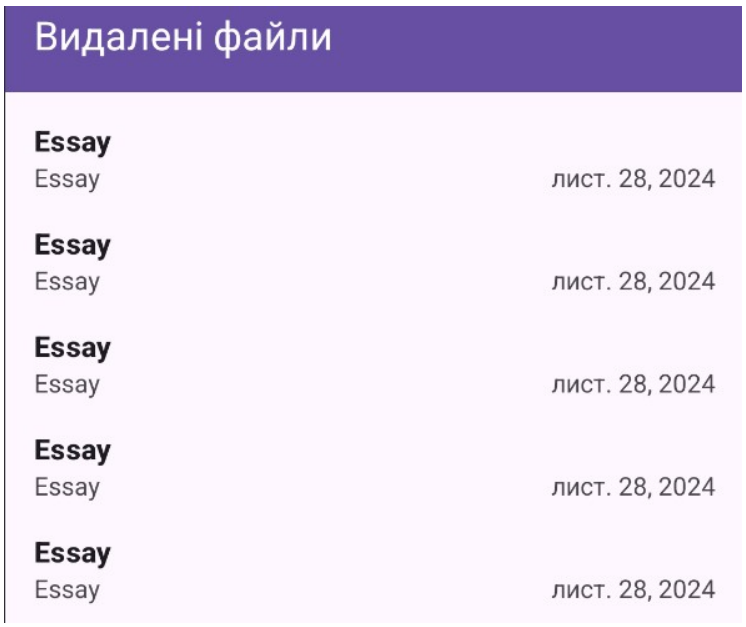

Рисунок 4.4 – Екран видалених файлів

Перейдемо до створення нового документа, повернувшись на головний екран застосунка, та натиснувши кнопку «+» в правій нижній його частині. З’являється діалогове вікно, в яке потрібно ввести назву файла, його тип, тему та призначення, як продемонстровано на рисунку 4.5.

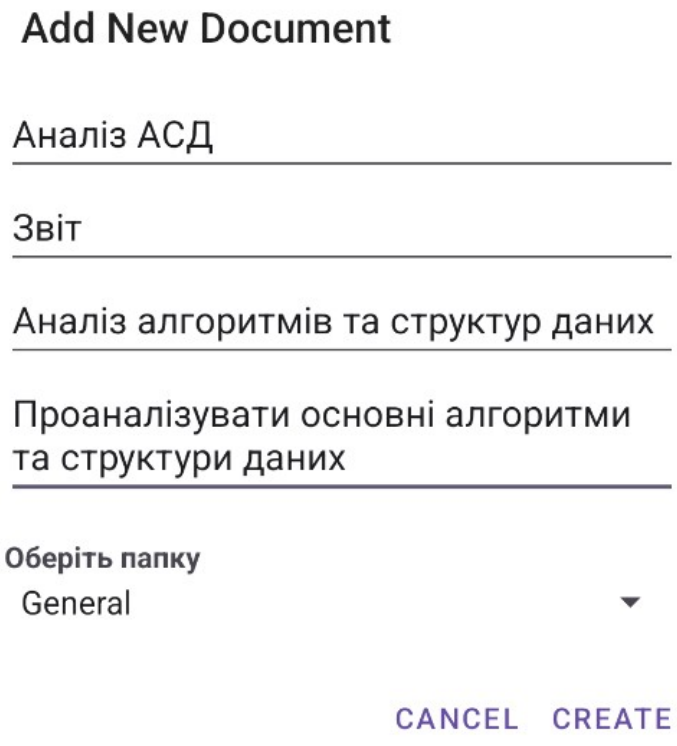

Рисунок 4.5 – Діалогове вікно створення нового документа

В нижній частині діалогового вікна присутній випадний список для вибору папки, в яку буде додано новостворений файл, як продемонстровано на рисунку 4.6.

Рисунок 4.6 – Меню вибору папки для файлу

Після натиснення кнопки «Створити», запускається панель прогресу, що обертається, для позначення того, що документ створюється (рисунок 4.7).

Рисунок 4.7 – Анімація завантаження під час налаштування структури документа та формування рекомендацій

Після того, як документ створено, діалогове вікно його створення закривається, та з’являється новий документ. Відкриємо цей документ. Можемо побачити структуру документа у вигляді розділів, як на рисунку 4.8.

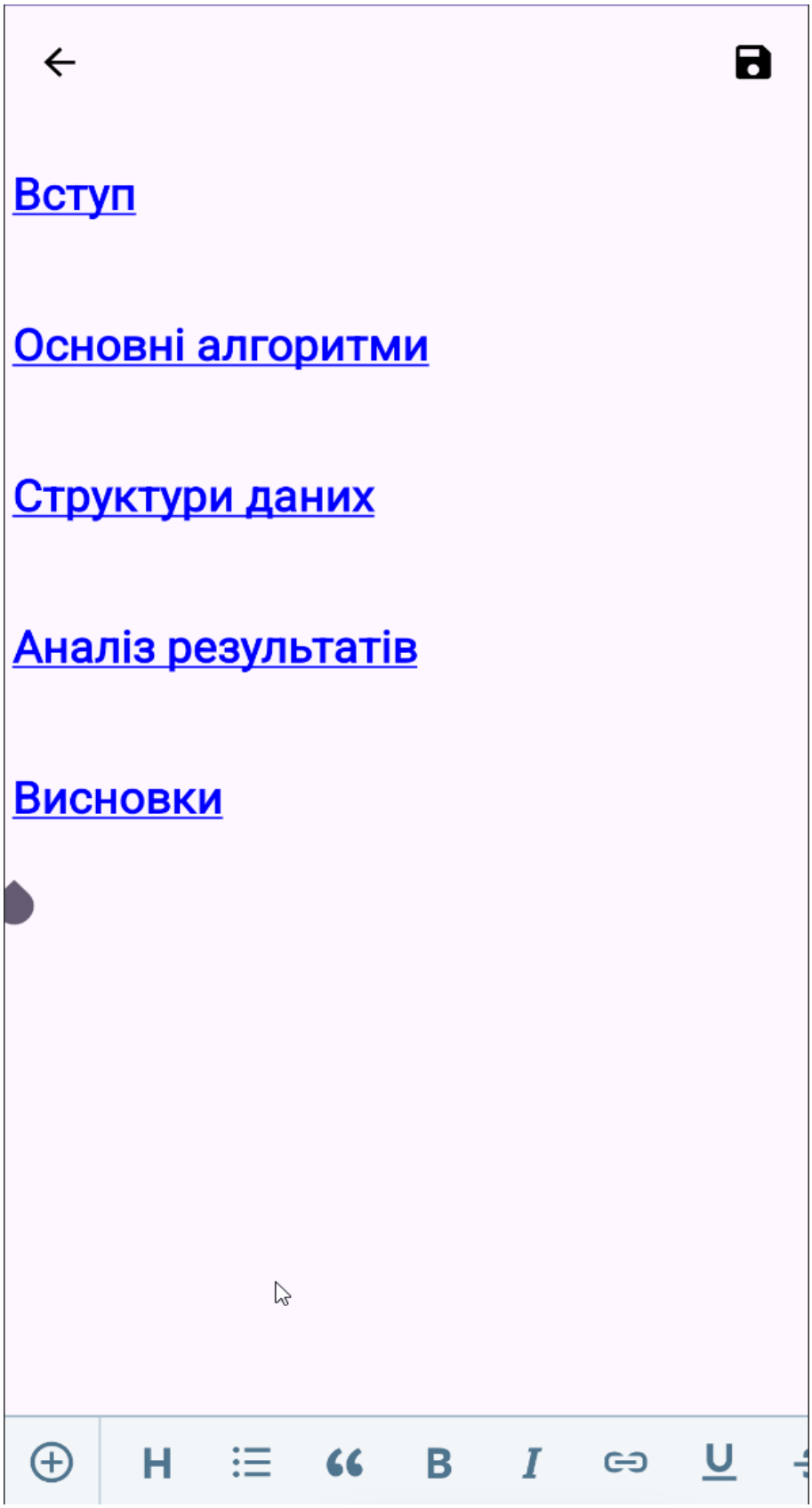

Рисунок 4.8 – Вікно редагування документа із завантаженою структурою

Усі розділи є клікабельними, натискання на них відкриває діалогове вікно

з переліком рекомендацій, що можна додати у цей документ (рисунок 4.9).

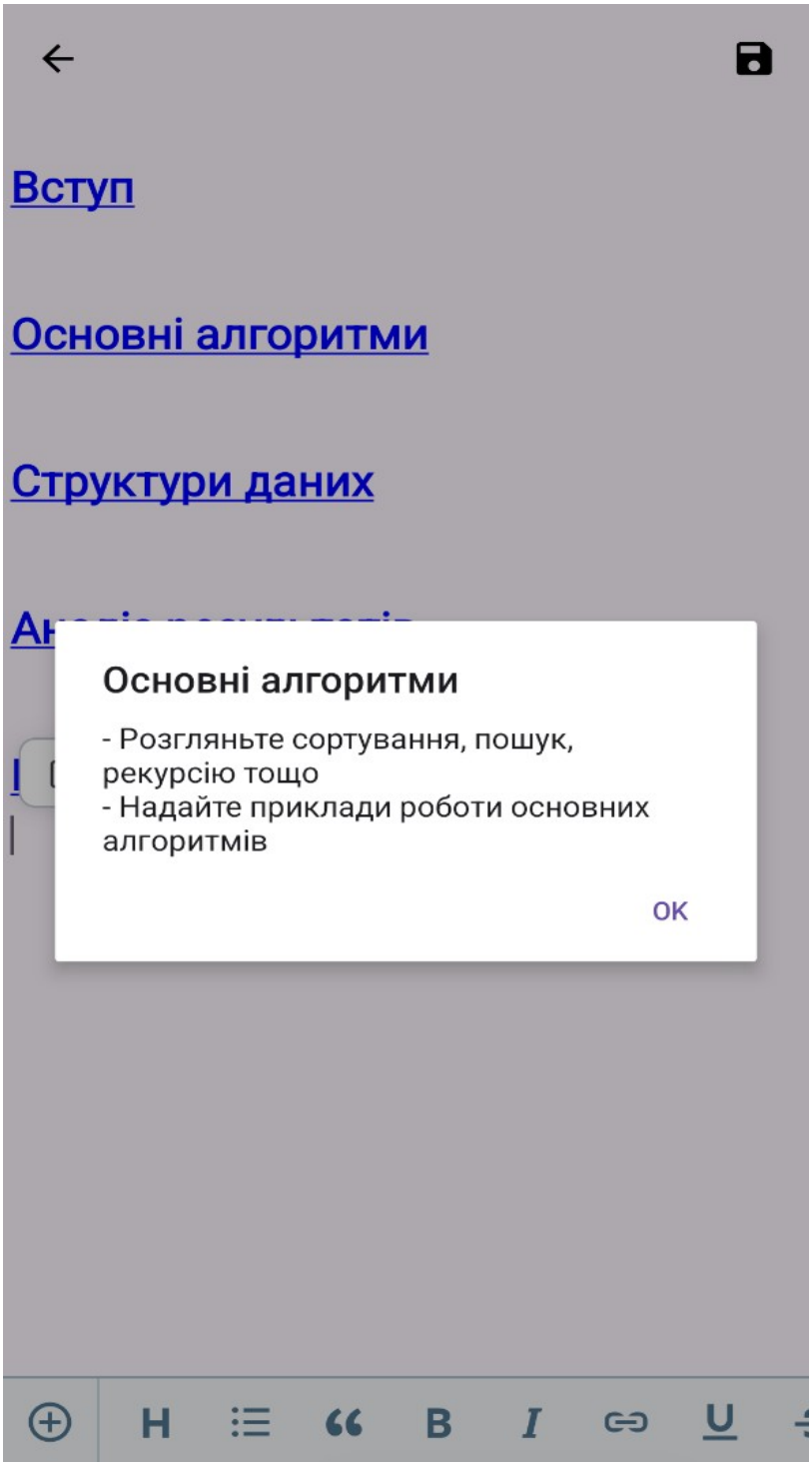

Рисунок 4.9 – Вікно з описом рекомендацій щодо наповнення обраного розділу документа

На рисунку 4.10 продемонстровано приклад заповнення документу текстом.

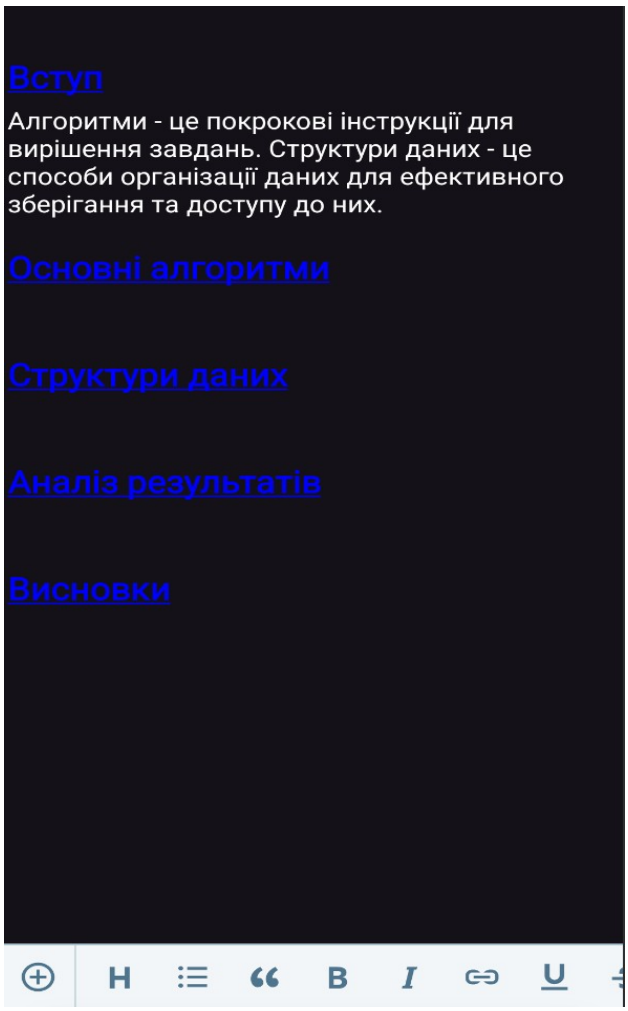

Рисунок 4.10 – Наповнення документа вмістом

Після введення тексту, система пропонує додати елемент візуалізації, зчитуючи доданий текст, дотримуючись отриманого контексту, як на рисунку 4.11.

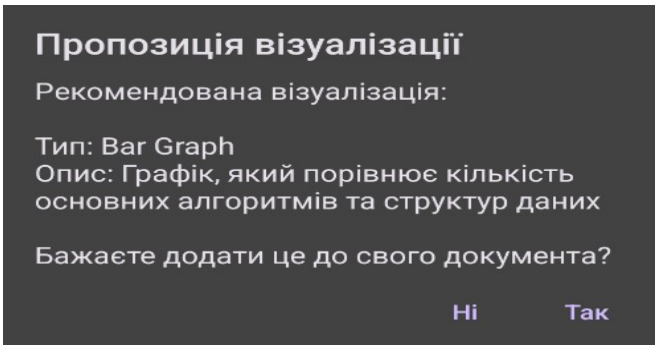

Рисунок 4.11 – Вікно з рекомендацією додати елемент візуалізації відповідно до контексту доданого тексту

Натиснемо «Так», після чого з’явиться діалогове вікно для додавання стовпчатої діаграми (рисунок 4.12).

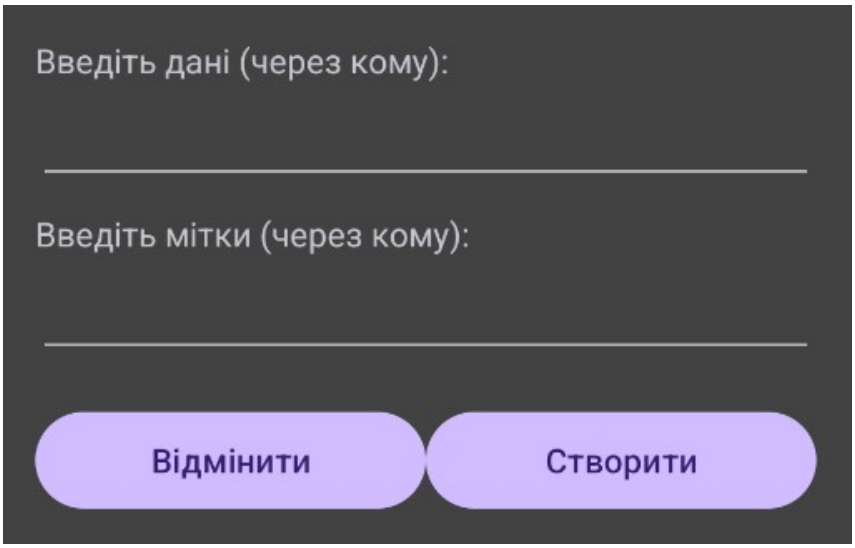

Рисунок 4.12 – Діалогове вікно створення діаграми

Введені дані у форму наведені на рисунку 4.13.

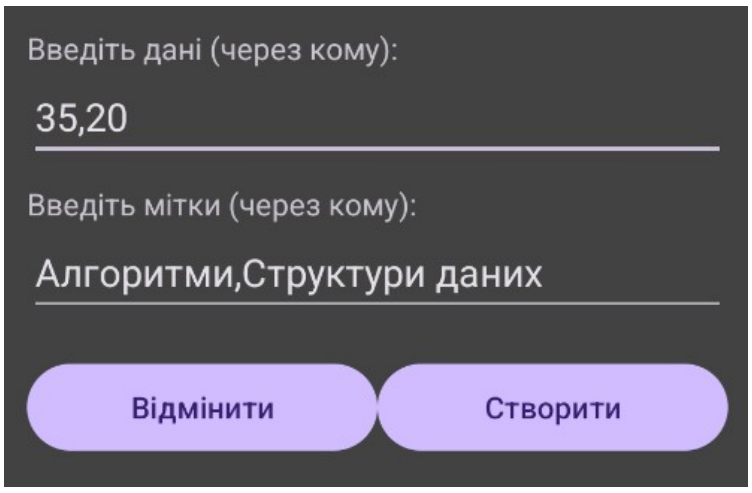

Рисунок 4.13 – Вікно створення діаграми із заповненими даними

На рисунку 4.14 продемонстровано створення та додавання стовпчатої діаграми до документа. Для цього, діаграма формується у вигляді зображення. При натисканні на нього, відкривається діалогове вікно з введеними даними, які можна редагувати, та оновити діаграму в документі. Таким же чином, окрім

стовпчатої діаграми, можна працювати із круговою, лінійною діаграмою, та таблицею.

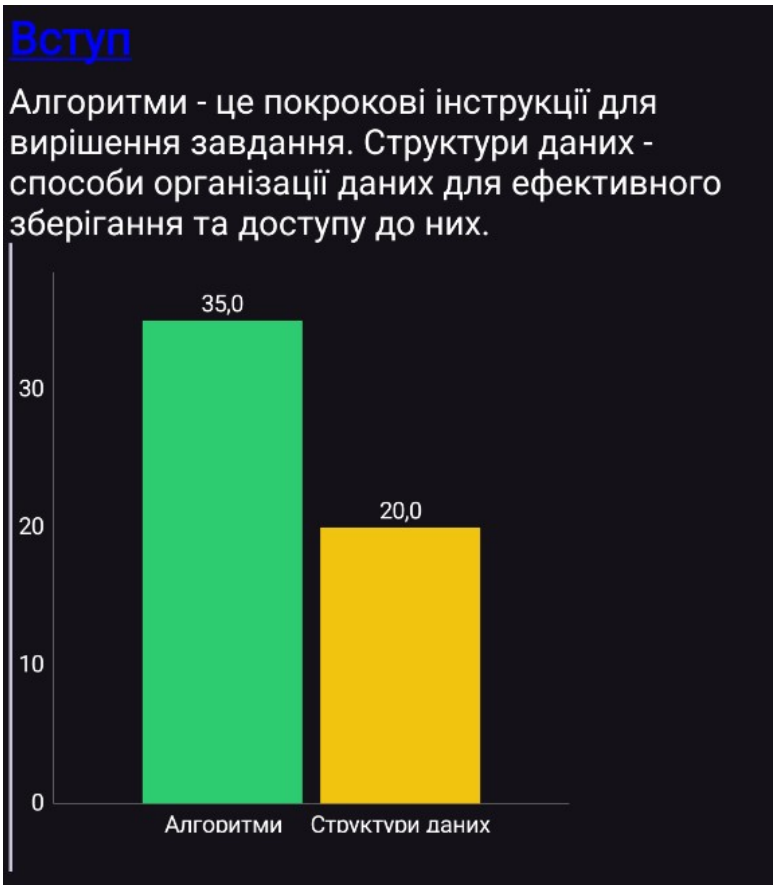

Рисунок 4.14 – Додана діаграма

Таким чином, було проведено тестування функціоналу програмних засобів системи для редагування й формування текстових документів. Було продемонстровано основний її функціонал та коректну його роботу.

4.3 Висновки

У четвертому розділі магістерської кваліфікаційної роботи, було проведено аналіз основних видів тестування. Обрано метод «чорної скриньки» для проведення тестування розроблених програмних засобів системи, обґрунтовано такий вибір. Проведено тестування програмних засобів системи для редагування й формування текстових документів. Продемонстровано його функціонал, коректну та очікувану роботу його функціоналу.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має сенс та може бути впроваджена лише за умови, що вона відповідає актуальним вимогам часу, підтримуючи науково-технічний прогрес і враховуючи економічні аспекти. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи).

Цей напрямок є пріоритетним, оскільки автоматизовані системи формування і редагування текстових документів підвищують ефективність та якість роботи, зменшують ризик помилок, відповідають сучасним тенденціям цифровізації, що сприяє конкурентоспроможності на ринку і задовольняє потреби користувачів.

Однак, задля реалізації цього проекту, необхідно довести його економічну доцільність, щоб знайти інвесторів, які будуть готові взяти участь у реалізації проекту.

Для наведеного випадку мають бути виконані такі етапи робіт:

1) комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації інвестором.

5.1 Проведення комерційного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за

темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 5.1 [31].

Таблиця 5.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі влас-тивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження таблиці 5.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь- які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки було зведено до таблиці 5.2.

Таблиця 5.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	3	4	4
2. Ринкові переваги (наявність аналогів)	3	3	3
3. Ринкові переваги (ціна продукту)	4	3	3
4. Ринкові переваги (технічні властивості)	4	4	4
5. Ринкові переваги (експлуатаційні витрати)	3	3	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	3	3	3
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	3	3	3
10. Практична здійсненність (необхідність нових матеріалів)	4	3	3
11. Практична здійсненність (термін реалізації)	4	4	4
12. Практична здійсненність (розробка документів)	4	4	4
Сума балів	42	41	41
Середньоарифметична сума балів $СБ_c$	41,3		

За результатами розрахунків, наведених в таблиці 5.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [32].

Таблиця 5.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ , розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» становить 41,3 бали, що, відповідно до таблиці 5.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

5.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам, конструкторам, технологам, креслярам, копіювальникам, лаборантам, робітникам, студентам, аспірантам та іншим працівникам, безпосередньо зайнятим виконанням конкретної теми, обчисленої за посадовими окладами, відрядними розцінками, тарифними ставками згідно з чинними в організаціях системами оплати праці.

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою 5.1 [32]:

$$З_о = \sum_{i=1}^{k} \frac{М_{пі} \cdot t_i}{Т_р}, \quad (5.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

$М_{пі}$ – місячний посадовий оклад конкретного дослідника, грн;

$t_і$ – число днів роботи конкретного дослідника, дні;

$Т_р$ – середнє число робочих днів в місяці, $Т_р$=22 дні.

$$З_о = 25000 \cdot 60 / 22 = 68181{,}81 \text{ грн.}$$

Проведені розрахунки зведено до таблиці 5.4.

Таблиця 5.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	25000	1136,36	60	68181,82
Інженер-розробник програмного забезпечення	20000	909,09	60	54545,45
Консультант	16000	727,27	30	21818,18
Дизайнер користувацького інтерфейсу	18000	818,18	20	16363,64
Всього				160909,09

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою 5.2:

$$З_р = \sum_{i=1}^{n} С_і \cdot t_i, \quad (5.2)$$

де $С_і$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою 5.3:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}}, \quad (5.3)$$

де M_M – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=8000,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду [31];

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дні;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 8000,00 \cdot 1,1 \cdot 1,65 / (22/8) = 82,5$ грн.

$З_{р1} = 82,5 \cdot 8,00 = 660$ грн.

Величина витрат на основну заробітну плату робітників наведена в табл.5.5.

Таблиця 5.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Підготовка робочого місця дослідника	8	2	1,1	82,5	660

Продовження таблиці 5.5

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
Інсталяція програмного забезпечення	3	5	1,7	127,5	382,5
Підключення апаратного забезпечення	8	3	1,35	101,25	810
Всього					1852,5

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою 5.4:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод} = (160909,09 + 1852,5) \cdot 11 / 100\% = 17903,78$ грн.

5.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою 5.5:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

$Зн = (160909,09 + 1852,5 + 17903,78) \cdot 22 / 100\% = 37946,38$ грн.

5.2.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою 5.6:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}, \quad (5.6)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 ... 1,15*);

B_j – маса відходів *j*-го найменування, кг;

$Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг.

$М_1 = 2 \cdot 179{,}00 \cdot 1{,}1 - 0{,}000 \cdot 0{,}00 = 393{,}8$ грн.

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір офісний Maestro Standard+ A4 80 г/м² B клас 500 аркушів	179	2	0	0	393,8
Канцелярське приладдя	250	3	0	0	825
Картридж для принтера Canon PG-440XL Black	974	1	0	0	1071,4

Продовження таблиці 5.6

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Флеш пам'ять USB Kingston DataTraveler Micro Gen2 64GB	369	1	0	0	405,9
Папка-бокс пластикова Economix A4 60 мм на гумці Синя	87	1	0	0	95,7
Всього					2791,8

5.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» відсутні.

5.2.5 Спецустаткування для наукових (експериментальних) робітника

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Витрати на спецустаткування, які використовують при проведенні НДР на тему «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» відсутні.

5.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних)

робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою 5.8:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \qquad (5.8)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг1} = 7899 \cdot 4 \cdot 1,1 = 43444,5$ *грн.*

Отримані результати зведемо до таблиці 5.7.

Таблиця 5.7 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	4	7899	34755,6
Прикладний пакет Microsoft Office 2021	4	9999	43995,6
Сервіс розробки інтерфейсів Figma	1	1800	1980
Всього			80731,2

5.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою 5.9:

$$A_{\text{обл}} = \frac{Ц_б}{T_в} \cdot \frac{t_{\text{вик}}}{12}, \quad (5.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (120000 \cdot 3) / (3 \cdot 12) = 10000$ *грн.*

Проведені розрахунки зведемо до таблиці 5.8.

Таблиця 5.8 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Персональний комп’ютер 4 шт	120000	3	3	10000
Робоче місце дослідника	16500	5	3	825
Оргтехніка	7500	4	3	468,75
ОС Windows 11 (4 шт.)	34755,6	2	3	2896,3
Прикладний пакет Microsoft Office 2021 (4 шт.)	43995,6	2	3	2444,3

Продовження таблиці 5.8

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Сервіс розробки інтерфейсів Figma	1980	1	3	495
Всього				17129,35

5.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою 5.10:

$$B_e = \sum_{i=1}^{n} \frac{W_{\text{уі}} \cdot t_i \cdot Ц_e \cdot K_{\text{впі}}}{\eta_i}, \qquad (5.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_е = 11$ грн;

$K_{впі}$ – коефіцієнт, що враховує використання потужності, $K_{впі} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 5 \cdot 0{,}25 \cdot 480{,}0 \cdot 11 \cdot 0{,}95 / 0{,}97 = 5171{,}13$ *грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Персональний комп’ютер (4 шт)	0,25	480	5171,13

Продовження таблиці 5.9

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Робоче місце дослідника	0,2	480	1034,23
Оргтехніка	0,4	16	68,95
Всього			6274,31

5.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою 5.11:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \quad (5.11)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (160909,09+ 1852,5) · 20 / 100% = 32552,32 грн.

5.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою 5.12:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$В_{сп}$ = (160909,09 + 1852,5) · 35 / 100% = 56966,56 грн.

5.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою 5.13:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.13)$$

де *$Н_{ів}$* – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 55%.

$І_в$ = (160909,09 + 1852,5) · 55 / 100% = 89518,88 грн.

5.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою 5.14:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 125%.

$В_{нзв}$ = (160909,09 + 1852,5) · 100 / 125% = 203451,99 грн.

Витрати на проведення науково-дослідної роботи розраховуємо як суму всіх попередніх статей витрат за формулою 5.15:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{смец} + В_{пргр} + А_{обл} + В_в + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.15)$$

$В_{заг}$=160909,09+1852,5+17903,78+39746,38+2791,8+0,00+80731,2+17129,35+ 6274,31+ 32552,32+56966,56+89518,88+203451,99 = 709828,16 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою 5.16:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,7.

ЗВ = 709828,16/ 0,7 = 1014040,23 грн.

5.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження передбачають вихід продукту на ринок протягом наступних трьох років. При цьому очікуваний економічний ефект буде ґрунтуватися на таких показниках:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що

аналізуються, від покращення його певних характеристик;

1-й рік – 4000 користувачів/рік;

2-й рік – 5000 користувачів/рік;

3-й рік – 7000 користувачів/рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 80000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 6000,00 грн /за рік користування;

$\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 500,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою 5.17 [31]:

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо $\rho = 30\%$;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1 =$ (80000·500,00+6000·4000)·0,83·0,3·(1-0,18/100%) = 13529458,8 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(80000·500,00+6000·(4000+5000))·0,83·0,3·(1-0,18/100%)= 18091651,31 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(80000·500,00+6000·(4000+5000+7000))·0,83·0,3·(1-0,18/100%)= 21335671,54 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою 5.18:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ=0,1;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$ПП$ = 13529458,8/(1+0,1)^1+ 18091651,31/(1+0,1)^2+ 21335671,54/(1+0,1)^3= 28510230,33 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки, розраховується за формулою 5.19:

$$PV = k_{інв} \cdot ЗВ, \quad (5.19)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=4;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 1014040,23 грн.

$PV = k_{інв} \cdot ЗВ$ = 4·1014040,23 = 4056160,92 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки розраховується за формулою 5.20:

$$E_{абс} = ПП - PV \tag{5.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 28510230,33 грн;

PV – теперішня вартість початкових інвестицій, 4056160,92 грн.

$E_{абс} = ПП - PV$ = 28510230,33 - 4056160,92 = 24454069,41 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.21:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 24454069,41 грн;

PV – теперішня вартість початкових інвестицій, 4056160,92 грн;

$T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її

впровадження, 3 роки.

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+24454069,41/4056160,92)^{1/3} -1 = 0,916.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$ розраховується за формулою 5.22:

$$\tau_{мін} = d + f, \quad (5.22)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,096;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,25.

$\tau_{мін}$ = 0,096+0,25 = 0,346 < 0,916 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою 5.23:

$$T_{ок} = \frac{1}{E_в}, \quad (5.23)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок}$ = 1 / 0,916 = 1,092 року.

$T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-

технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів» становить 41,3 бали, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 1,092 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів».

ВИСНОВКИ

У результаті виконання магістерської кваліфікаційної роботи було розроблено методи та програмні засоби системи автоматизованого редагування й формування текстових документів. Робота оформлена згідно методичних вказівок [33, 34].

Було проведено аналіз предметної області та стану систем для автоматизованого редагування й формування текстових документів, порівняльний аналіз аналогів, в результаті якого було доведено актуальність власної розробки. Проведено аналіз методів розв’язання задачі.

Розроблено систему для автоматизованого редагування й формування текстових документів, яка включає функціонал для автоматичної генерації структури документа, формування рекомендацій щодо наповнення документа вмістом, формування рекомендацій щодо елементів візуалізації на основі введеного користувачем тексту, додавання таблиць, зображень та графіків до документа.

Розроблено метод формування структури документа, який використовує штучний інтелект для формування розділів та структурних частин документа, враховуючи його тему та призначення.

Розроблено метод рекомендацій наповнення документа, який використовує штучний інтелект для надання точних рекомендацій залежно від теми та призначення документа.

Розроблено метод рекомендацій елементів візуалізації, який відстежує зміни документа та додає елементи візуалізації для підвищення інформативності документа та згідно з його контекстом.

Розроблено метод роботи з елементами візуалізації у Android View інтерфейсі текстового редактора, який використовує діалогові вікна та конвертацію елементів візуалізації у зображення.

Проведено аналіз методів тестування. Тестування системи довело її працездатність та відповідність поставленим вимогам.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Physical appearance of documents [Електронний ресурс] – Режим доступу до ресурсу: https://www.britannica.com/topic/diplomatics/Physical-appearance-of-documents

2. Музичук Д.Р. Розробка засобів системи автоматизованого формування і редагування текстових документів/ Д.Р. Музичук, В.В. Войтко, Г.О.Черноволик //Інформаційні технології і автоматизація – 2024 / Матеріали XVII міжнародної науково-практичної конференції. Одеса, 31 жовтня - 1 листопада 2024 р. - Одеса, Видавництво ОНТУ, 2024 р. – 479-482 с.

3. Ельперін І.В. Автоматизація виробничих процесів. /І. В. Ельперін, О. М. Пупена, В. М. Сідлецький, С. М. Швед. Київ: Ліра-К, 2021. 378с.

4. Microsoft Word [Електронний ресурс] – Режим доступу до ресурсу:

5. Formstack Documents [Електронний ресурс] – Режим доступу до ресурсу: https://www.formstack.com/products/documents

6. Docmosis [Електронний ресурс] – Режим доступу до ресурсу: https://www.docmosis.com/

7. Jasper [Електронний ресурс] – Режим доступу до ресурсу: https://www.jasper.ai/

8. Fathi Saad. A Classification Method to Extract Knowledge from Text Documents. Лондон: LAP LAMBERT Academic Publishing, 2022. 220с.

9. Pieter Nijs. The MVVM Pattern in .NET MAUI. Бірмінґем: Packt Publishing, 2023. 386с.

10. Mark Seemann. Dependency Injection Principles, Practices, and Patterns. Нью-Йорк: Manning Publications, 2019. 552с.

11. Massimo Carli. Dagger by Tutorials. Мак-Гахейсвілл: Razeware LLC, 2021. 544с.

12. Розробка мобільних додатків на MVVM [Електронний ресурс] – Режим доступу до ресурсу: https://brander.ua/technologies/mvvm

13. Kotlin Programming Language [Електронний ресурс] – Режим доступу

до ресурсу: https://kotlinlang.org/

14. Kotlin and Android [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/kotlin

15. Dart [Електронний ресурс] – Режим доступу до ресурсу: https://dart.dev

16.Dr. Deepti Chopra. Flutter and Dart: Up and Running. Нойда: BPB Publications, 2023. 200с.

17. David Flanagan. JavaScript. The Definitive Guide. Себастопол: O'Reilly, 2020. 706с.

18. Angular [Електронний ресурс] – Режим доступу до ресурсу: https://angular.dev/

19. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

20. Joseph Albahari. C# 12 in a Nutshell. Себастопол: O'Reilly, 2023. 1083с.

21. C# [Електронний ресурс] – Режим доступу до ресурсу: https://infdev.com.ua/docs/%D1%81oding-languages/csharp/

22. IntelliJ IDEA [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/idea/

23. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio

24. AppCode [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/objc/

25. Android View[Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/reference/android/view/View

26. Thomas Künneth. Android UI Development with Jetpack Compose. Бірмінґем: Packt Publishing, 2022. 248с.

27. Mauricio Aniche. Effective Software Testing: A developer's guide. Нью-Йорк: Manning, 2022. 328с.

28. Erik van Veenendaal. Foundations of Software Testing ISTQB Certification. Бостон: Cengage Learning EMEA, 2019. 288с.

29. Ash Winter. Team Guide to Software Testability. Лідс: Conflux Books, 2021. 188с.

30. Lee Copeland. A Practitioner's Guide to Software Test Design. Лондон: Artech House, 2004. 312с.

31.Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

33.Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк. Вінниця : ВНТУ, 2021. – 60 с.

34. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

ДОДАТКИ

113

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«19» вересня 2024 р.

Технічне завдання
на магістерську кваліфікаційну роботу «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
к.т.н., доц. В.В. Войтко
«19» вересня 2024 року.

Виконав:
студент гр. 2ПІ-23м Д.Р. Музичук
«19» вересня 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів».

Галузь застосування – мобільні системи.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ №310 від 17 вересня 2024 р. ректора ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є підвищення рівня автоматизації процесу розробки текстових документів шляхом розробки та впровадження системи автоматизованого формування і редагування документів, що дозволить підвищити якість та надійність обробки текстових файлів.

4. Вихідні дані для проведення НДР

1. Розробка мобільних додатків на MVVM [Електронний ресурс] – Режим доступу до ресурсу: htttps://brander.ua/technologies/mvvm

2. Kotlin Programming Language [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/

3. Методичні вказівки до виконання магістерської кваліфікаційної роботи для студентів спеціальності 121 «Інженерія програмного забезпечення» / уклад. : О. Н. Романюк, Г. О. Черноволик. – Вінниця : ВНТУ, 2022. – 50 с.

5. Технічні вимоги

Вхідні дані – користувач системи, документи, папки, створені документи.

Вихідні дані – графічний інтерфейс користувача з можливістю створення та редагування текстових документів.

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 МКР;
- 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 магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану систем для редагування й формування текстових документів	20.09.2024 30.09.2024
2	Розробка методів, моделі та алгоритмів роботи системи	01.10.2024 17.10.2024
3	Розробка програмних засобів системи	18.10.2024 04.11.2024
4	Тестування системи	05.11.2024 21.11.2024
5	Економічна частина	22.11.2024 30.11.2024

10. Порядок контролю та прийняття.

Усі етапи магістерської кваліфікаційної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

116

ДОДАТОК Б

Протокол перевірки на плагіат

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: **Розробка методу та засобів системи автоматизованого формування і редагування текстових документів.**

Тип роботи: кваліфікаційна робота
Підрозділ : кафедра програмного забезпечення, ФІТКІ, 2ПІ-23м
Науковий керівник: к.т.н., доц. каф. ПЗ Войтко В.В.

Turnitin	
Оригінальність	96 %
Схожість	4 %

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.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи «Розробка методу та засобів системи автоматизованого формування і редагування текстових документів».

Автор _____ Музичук Дмитро Романович

Опис прийнятого рішення: **допустити до захисту**

Особа, відповідальна за перевірку _____ Черноволик Г. О.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)

ДОДАТОК В

Лістинг коду

MainActivity.kt

```
@AndroidEntryPoint
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    // DrawerLayout and NavigationView
    private lateinit var drawerLayout: DrawerLayout
    private lateinit var navigationView: NavigationView

    // Toolbar Views
    private lateinit var searchView: SearchView
    private lateinit var buttonSort: ImageButton
    private lateinit var buttonDrawer: ImageButton
    private lateinit var toolbar: Toolbar

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // Initialize view binding
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Initialize DrawerLayout and NavigationView
        drawerLayout = binding.drawerLayout
        navigationView = binding.navigationView

        // Initialize Toolbar and its components
        toolbar = binding.toolbar
        searchView = toolbar.findViewById(R.id.searchView)
        buttonSort = toolbar.findViewById(R.id.buttonSort)
        buttonDrawer = toolbar.findViewById(R.id.buttonDrawer)

        // Set up the toolbar
        setSupportActionBar(toolbar)
        supportActionBar?.setDisplayShowTitleEnabled(false) // Remove default title text

        // Handle navigation drawer button click
        buttonDrawer.setOnClickListener {
            drawerLayout.openDrawer(GravityCompat.START)
        }

        // Set up NavigationView item selection listener
        navigationView.setNavigationItemSelectedListener { menuItem ->
            when (menuItem.itemId) {
```

```
            R.id.nav_all_files -> {
                navigateToDocumentsFragment()
                drawerLayout.closeDrawer(GravityCompat.START)
                true
            }
            R.id.nav_deleted_files -> {
                navigateToDeletedFilesFragment()
                drawerLayout.closeDrawer(GravityCompat.START)
                true
            }
            R.id.nav_manage_folders -> {
                navigateToManageFoldersFragment()
                drawerLayout.closeDrawer(GravityCompat.START)
                true
            }
            else -> false
        }
    }

    // Set up SearchView
    setupSearchView()

    // Set up Sort Button
    buttonSort.setOnClickListener {
        showSortOptions()
    }

    // Load the default fragment
    if (savedInstanceState == null) {
        navigateToDocumentsFragment()
    }
}

private fun navigateToDocumentsFragment() {
    val fragment = DocumentsFragment()
    supportFragmentManager.beginTransaction()
        .replace(R.id.fragment_container, fragment)
        .commit()
}

private fun navigateToDeletedFilesFragment() {
    val fragment = DeletedFilesFragment()
    supportFragmentManager.beginTransaction()
        .replace(R.id.fragment_container, fragment)
        .commit()
}

private fun navigateToManageFoldersFragment() {
    val fragment = ManageFoldersFragment()
    supportFragmentManager.beginTransaction()
        .replace(R.id.fragment_container, fragment)
        .commit()
}
```

```
override fun onBackPressed() {
    if (drawerLayout.isDrawerOpen(GravityCompat.START)) {
        drawerLayout.closeDrawer(GravityCompat.START)
    } else {
        super.onBackPressed()
    }
}

private fun setupSearchView() {
    searchView.setOnQueryTextListener(object : SearchView.OnQueryTextListener {
        override fun onQueryTextSubmit(query: String?): Boolean {
            // Handle search query submission
            query?.let {
                performSearch(it)
            }
            return true
        }

        override fun onQueryTextChange(newText: String?): Boolean {
            // Handle text changes in the search bar
            newText?.let {
                performSearch(it)
            }
            return true
        }
    })
}



enum class SortOption {
    NAME,
    DATE_CREATED,
    TYPE
}

private fun showSortOptions() {
    // Show sorting options
    val options = arrayOf("Name", "Date Created", "Type")
    val builder = AlertDialog.Builder(this)
    builder.setTitle("Sort By")
    builder.setItems(options) { _, which ->
        when (which) {
            0 -> performSort(SortOption.NAME)
            1 -> performSort(SortOption.DATE_CREATED)
            2 -> performSort(SortOption.TYPE)
        }
    }
    builder.show()
}
```

MainViewModel.kt

```
@HiltViewModel
class MainViewModel @Inject constructor(
    private val documentDao: DocumentDao ,
    private val folderDao: FolderDao ,
    private val openAIClient: OpenAIClient = OpenAIClient()
) : ViewModel() {

    private val _searchQuery = MutableLiveData<String>("")
    private val _currentFolderId = MutableLiveData<String?>(null)
    private val _showDeleted = MutableLiveData<Boolean>(false)
    private val _sortCriteria = MutableLiveData<String>("name")

    val allFolders: LiveData<List<Folder>> = folderDao.getAllFolders()

    val allDocuments: LiveData<List<Document>> = documentDao.getAllDocuments()

    // LiveData for documents
    private val _documents = MutableLiveData<List<Document>>()
    val documents: LiveData<List<Document>> get() = _documents

    // UI State (if applicable)
    private val _uiState = MutableLiveData<UiState>(UiState())
    val uiState: LiveData<UiState> get() = _uiState

    private val _displayedDocuments = MediatorLiveData<List<Document>?>()
    val displayedDocuments: MediatorLiveData<List<Document>?> get() = _displayedDocuments

    val deletedDocuments: LiveData<List<Document>> = documentDao.getDeletedDocuments()

    private val _createdDocument = MutableLiveData<Document?>()
    val createdDocument: LiveData<Document?> get() = _createdDocument

    private val _visualizationSuggestion = MutableLiveData<VisualizationSuggestion?>()
    val visualizationSuggestion: LiveData<VisualizationSuggestion?> = _visualizationSuggestion

    private val languageCode: String = Locale.getDefault().language

    private val language = when(languageCode){
        "uk"->"українською"
        else -> "English"
    }

    fun analyzeTextForVisualization(text: String) {
        val textToAnalyze = text.takeLast(1000) // Adjust as needed

        viewModelScope.launch {
            try {
                val prompt = buildVisualizationPrompt(textToAnalyze)
```

```
                val response = openAIClient.callOpenAIApi(prompt)
                val suggestion = handleVisualizationResponse(response)
                _visualizationSuggestion.postValue(suggestion)
            } catch (e: Exception) {
                e.printStackTrace()
                _visualizationSuggestion.postValue(null)
            }
        }
    }

    private fun buildVisualizationPrompt(text: String): String {
        return """
            As a document assistant, analyze the following text and suggest in $language language specific visualization elements that would enhance the information presented. Provide your suggestions in the following JSON format:

            {
                "type": "Visualization Type (e.g., 'Pie Chart', 'Bar Graph', 'Line Chart', 'Image')",
                "description": "A brief description of the visualization",
                "data": "If applicable, the data points or variables to visualize, separated by commas"
            }

            Ensure that the JSON is properly formatted.

            Text:
            "$text"
        """.trimIndent()
    }

    data class VisualizationSuggestion(
        val type: String,
        val description: String,
        val data: String?
    )

    private fun handleVisualizationResponse(response: String): VisualizationSuggestion? {
        try {
            val suggestion = parseVisualizationResponse(response)
            return suggestion
        } catch (e: Exception) {
            e.printStackTrace()
            return null
        }
    }

    private fun parseVisualizationResponse(response: String): VisualizationSuggestion? {
        val gson = Gson()
        val jsonString = extractJson(response)
        try {
            val suggestion = gson.fromJson(jsonString, VisualizationSuggestion::class.java)
            return suggestion
        } catch (e: JsonSyntaxException) {
            e.printStackTrace()
            return null
```

```
    }
}

private fun extractJson(response: String): String {
    val jsonStart = response.indexOf("{")
    val jsonEnd = response.lastIndexOf("}")
    if (jsonStart != -1 && jsonEnd != -1 && jsonEnd >= jsonStart) {
        return response.substring(jsonStart, jsonEnd + 1)
    }
    throw JSONException("Invalid JSON in response")
}




val filteredAndSortedDocuments: LiveData<List<Document>> = MediatorLiveData<List<Document>>().apply {
    val update = {
        val query = _searchQuery.value ?: ""
        val criteria = _sortCriteria.value ?: "name"
        val liveData = if (query.isEmpty()) {
            getDocumentsSorted(criteria)
        } else {
            getFilteredAndSortedDocuments(query, criteria)
        }
        addSource(liveData) { documents ->
            value = documents
        }
    }

    addSource(_searchQuery) { update() }
    addSource(_sortCriteria) { update() }
}

private fun getDocumentsSorted(criteria: String): LiveData<List<Document>> {
    return when (criteria) {
        "name" -> documentDao.getDocumentsSortedByName()
        "date" -> documentDao.getDocumentsSortedByDate()
        // Add other criteria as needed
        else -> documentDao.getAllDocuments()
    }
}

private fun getFilteredAndSortedDocuments(query: String, criteria: String): LiveData<List<Document>> {
    val formattedQuery = "%$query%"
    return when (criteria) {
        "name" -> documentDao.searchDocumentsSortedByName(formattedQuery)
        "date" -> documentDao.searchDocumentsSortedByDate(formattedQuery)
        // Add other criteria as needed
        else -> documentDao.searchDocuments(formattedQuery)
    }
}
```

```
fun saveDocument(document: Document) {
    viewModelScope.launch {
        documentDao.updateDocument(document)
    }
}

fun deleteDocument(document: Document) {
    viewModelScope.launch {
        documentDao.updateDocument(document)
    }
}

fun restoreDocument(document: Document) {
    viewModelScope.launch {
        val restoredDocument = document.copy(deleted = false)
        documentDao.updateDocument(restoredDocument)
    }
}

fun createDocument(name: String, type: String, topic: String, purpose: String, folderId: String? = null) {
    _uiState.value = _uiState.value?.copy(isLoading = true, errorMessage = null)

    viewModelScope.launch {
        try {
            val sections = generateSectionsAndTips(type, topic, purpose)
            val document = Document(
                id = UUID.randomUUID().toString(),
                name = name,
                type = type,
                topic = topic,
                purpose = purpose,
                sections = sections,
                folderId = folderId,
                dateCreated = System.currentTimeMillis(),
                deleted = false
            )
            // Save to database
            documentDao.insertDocument(document)

            _createdDocument.postValue(document)

            // Update UI state
            _uiState.value = _uiState.value?.copy(
                recentDocuments = listOf(document) + (_uiState.value?.recentDocuments ?: emptyList()),
                currentDocument = document,
                showDocumentEditor = true,
                showNewDocumentDialog = false,
                isLoading = false
            )
        } catch (e: Exception) {
            _uiState.value = _uiState.value?.copy(
                isLoading = false,
```

```
                    errorMessage = e.message
                )
            }
        }
    }

    fun resetCreatedDocument() {
        _createdDocument.value = null
    }


    fun getDocumentsInFolder(folderId: String): LiveData<List<Document>> {
        return documentDao.getDocumentsInFolder(folderId)
    }

    private suspend fun generateSectionsAndTips(
        type: String,
        topic: String,
        purpose: String
    ): List<Section> {


        val prompt = buildPrompt(type, topic, purpose)
        val response = openAIClient.callOpenAIApi(prompt)
        return parseSectionsFromResponse(response)
    }

    private fun buildPrompt(type: String, topic: String, purpose: String): String {
        return """
            Як помічник експерта з документів, створіть структурований план для "$type" на тему "$topic" із призначенням "$purpose" $language мовою.
            Для кожного розділу надайте:
            - Назва
            - 2-3 поради, що включити
            Надайте **тільки** результат у форматі масиву JSON без додаткового тексту, без використання маркерів коду або додаткового форматування.
            Масив JSON має складатися з об’єктів із полями: «title», «tips».
        """.trimIndent()
    }

    private fun parseSectionsFromResponse(response: String): List<Section> {
        val gson = Gson()

        // Extract JSON from the assistant's response
        val jsonStartIndex = response.indexOf('[')
        val jsonEndIndex = response.lastIndexOf(']') + 1

        if (jsonStartIndex != -1 && jsonEndIndex != -1 && jsonEndIndex > jsonStartIndex) {
            val jsonContent = response.substring(jsonStartIndex, jsonEndIndex)
            // Parse the JSON content into SectionResponse
            try {
                val listType = object : TypeToken<List<SectionResponse>>() {}.type
                val sectionResponses: List<SectionResponse> = gson.fromJson(jsonContent, listType)
```

```
            // Map SectionResponse to Section
            val sections = sectionResponses.mapIndexed { index, sectionResponse ->
                Section(
                    id = index + 1,
                    title = sectionResponse.title,
                    tips = sectionResponse.tips,
                    // Initialize other fields with default values
                    contentType = ContentType.TEXT,
                    content = ""
                )
            }
            return sections
        } catch (e: JsonSyntaxException) {
            println("Error parsing sections: ${e.message}")
            println("JSON Content: $jsonContent")
            throw e
        }
    } else {
        println("Could not find JSON array in the assistant's response.")
        println("Assistant's Response: $response")
        throw Exception("Could not find JSON array in the assistant's response.")
    }
}


fun getDocument(documentId: String): LiveData<Document> {
    return documentDao.getDocumentById(documentId)
}

fun updateDocument(document: Document) {
    viewModelScope.launch {
        documentDao.updateDocument(document)
    }
}

@kotlinx.serialization.Serializable
data class SectionResponse(
    val title: String,
    val tips: List<String>,
    val recommendedInteractiveElements: List<String>
)

data class UiState(
    val isLoading: Boolean = false,
    val errorMessage: String? = null,
    val recentDocuments: List<Document> = emptyList(),
    val currentDocument: Document? = null,
    val showDocumentEditor: Boolean = false,
    val showNewDocumentDialog: Boolean = false
)
}
```

EditorActivity.kt

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_editor)

    binding = ActivityEditorBinding.inflate(layoutInflater)

    editor = binding.aztec

    setupBackButton()
    document = intent.getParcelableExtra(EXTRA_DOCUMENT)

    testImageView = findViewById(R.id.testImageView)


    setContentView(binding.root)


    onBackPressedDispatcher.addCallback(this , object : OnBackPressedCallback(true) {
      override fun handleOnBackPressed() {
        mIsKeyboardOpen = false
        showActionBarIfNeeded()

        isEnabled = false
        onBackPressedDispatcher.onBackPressed()
        isEnabled = true
      }
    })

    // Setup hiding the action bar when the soft keyboard is displayed for narrow viewports
    if (resources.configuration.orientation == Configuration.ORIENTATION_LANDSCAPE
      && !resources.getBoolean(R.bool.is_large_tablet_landscape)
    ) {
      mHideActionBarOnSoftKeyboardUp = true
    }

    val visualEditor = findViewById<AztecText>(R.id.aztec)
    val sourceEditor = findViewById<SourceViewEditText>(R.id.source)
    aztecToolbar = findViewById(R.id.formatting_toolbar)

    visualEditor.enableSamsungPredictiveBehaviorOverride()

    visualEditor.externalLogger = object : AztecLog.ExternalLogger {
      override fun log(message: String) {
      }

      override fun logException(tr: Throwable) {
      }

      override fun logException(tr: Throwable , message: String) {
```

```
    }
}

val galleryButton = MediaToolbarGalleryButton(aztecToolbar)
galleryButton.setMediaToolbarButtonClickListener(object :
    IMediaToolbarButton.IMediaToolbarClickListener {
    override fun onClick(view: View) {
        mediaMenu = PopupMenu(this@EditorActivity , view)
        mediaMenu?.setOnMenuItemClickListener(this@EditorActivity)
        mediaMenu?.inflate(R.menu.menu_gallery)
        mediaMenu?.show()
        if (view is ToggleButton) {
            view.isChecked = false
        }
    }
})

val cameraButton = MediaToolbarCameraButton(aztecToolbar)
cameraButton.setMediaToolbarButtonClickListener(object :
    IMediaToolbarButton.IMediaToolbarClickListener {
    override fun onClick(view: View) {
        mediaMenu = PopupMenu(this@EditorActivity , view)
        mediaMenu?.setOnMenuItemClickListener(this@EditorActivity)
        mediaMenu?.inflate(R.menu.menu_camera)
        mediaMenu?.show()
        if (view is ToggleButton) {
            view.isChecked = false
        }
    }
})


val tableButton = MediaToolbarTableButton(aztecToolbar)
tableButton.setMediaToolbarButtonClickListener(object : IMediaToolbarButton.IMediaToolbarClickListener {
    override fun onClick(view: View) {
        showTableDialog()
    }
})

val saveButton = findViewById<ImageButton>(R.id.save_button)
saveButton.setOnClickListener {
    saveDocument()
}

aztec = Aztec.with(visualEditor , sourceEditor , aztecToolbar , this)
    .setImageGetter(GlideImageLoader(this))
    .setVideoThumbnailGetter(GlideVideoThumbnailLoader(this))
    .setOnImeBackListener(this)
    .setOnTouchListener(this)
    .setHistoryListener(this)
    .setOnImageTappedListener(this)
    .setOnVideoTappedListener(this)
    .setOnAudioTappedListener(this)
```

```
            .addOnMediaDeletedListener(this)
            .setOnVideoInfoRequestedListener(this)
            .addPlugin(WordPressCommentsPlugin(visualEditor))
            .addPlugin(MoreToolbarButton(visualEditor))
            .addPlugin(PageToolbarButton(visualEditor))
            .addPlugin(CaptionShortcodePlugin(visualEditor))
            .addPlugin(VideoShortcodePlugin())
            .addPlugin(AudioShortcodePlugin())
            .addPlugin(HiddenGutenbergPlugin(visualEditor))
            .addPlugin(galleryButton)
            .addPlugin(cameraButton)
            .addPlugin(tableButton)
            .addPlugin(initializePieChartButton())
            .addPlugin(initializeBarChartButton())
            .addPlugin(initializeLineChartButton())

//          initializeCustomButtons()

        document = intent.getParcelableExtra(EXTRA_DOCUMENT)
        if (document == null) {
            Toast.makeText(this , "Failed to load document." , Toast.LENGTH_SHORT).show()
            finish()
            return
        }

        // Convert sections to HTML
        val htmlContent = sectionsToHtml(document!!.sections)

        // Load the content into the editor
        binding.aztec.fromHtml(htmlContent)

        // Observe visualization suggestions from ViewModel
        viewModel.visualizationSuggestion.observe(this , { suggestion ->
            suggestion?.let {
                showVisualizationRecommendationDialog(it)
            }
        })

        // Add TextWatcher to Aztec editor
        binding.aztec.addTextChangedListener(object : TextWatcher {
            private var handler = Handler(Looper.getMainLooper())
            private var runnable: Runnable? = null
            private val debounceDelay = 2000L // Adjust as needed

            override fun beforeTextChanged(
                s: CharSequence? ,
                start: Int ,
                count: Int ,
                after: Int
            ) {
            }

            override fun onTextChanged(s: CharSequence? , start: Int , before: Int , count: Int) {}
```

```
    override fun afterTextChanged(s: Editable?) {
        runnable?.let { handler.removeCallbacks(it) }
        runnable = Runnable {
            s?.let {
                // Pass the text to ViewModel for analysis
                viewModel.analyzeTextForVisualization(it.toString())
            }
        }
        handler.postDelayed(runnable!! , debounceDelay)
    }
})

// Set up clickable section titles
setupClickableSectionTitles()

aztec.visualEditor.movementMethod = LinkMovementMethod.getInstance()
aztec.visualEditor.setOnImageTappedListener(this)
aztec.visualEditor.setOnMediaDeletedListener(this)


// initialize the plugins, text & HTML
if (!isRunningTest) {
    aztec.visualEditor.enableCrashLogging(object :
        AztecExceptionHandler.ExceptionHandlerHelper {
        override fun shouldLog(ex: Throwable): Boolean {
            return true
        }
    })
    aztec.visualEditor.setCalypsoMode(false)
    aztec.sourceEditor?.setCalypsoMode(false)

    aztec.visualEditor.setBackgroundSpanColor(
        ContextCompat.getColor(
            this ,
            AztecR.color.blue_dark
        )
    )


    aztec.addPlugin(CssUnderlinePlugin())
    aztec.addPlugin(CssBackgroundColorPlugin())
    aztec.addPlugin(BackgroundColorButton(visualEditor))
}

if (savedInstanceState == null) {

    aztec.initSourceEditorHistory()
}

invalidateOptionsHandler = Handler(Looper.getMainLooper())
invalidateOptionsRunnable = Runnable { invalidateOptionsMenu() }
```

```
    }
```

CreateDocumentDialogFragment.kt

```
class CreateDocumentDialogFragment : DialogFragment() {

    private lateinit var viewModel: MainViewModel
    private var _binding: DialogCreateDocumentBinding? = null
    private val binding get() = _binding!!

    // Adapter for the Spinner
    private lateinit var folderAdapter: ArrayAdapter<String>
    private val folderNameList = mutableListOf<String>()
    private val folderIdList = mutableListOf<String?>() // Nullable for "No Folder"

    override fun onCreateDialog(savedInstanceState: Bundle?): AlertDialog {
        // Initialize ViewModel
        viewModel = ViewModelProvider(requireActivity()).get(MainViewModel::class.java)

        // Inflate the layout using View Binding
        _binding = DialogCreateDocumentBinding.inflate(layoutInflater)

        // Build the dialog
        val dialog = AlertDialog.Builder(requireContext())
            .setTitle("Add New Document")
            .setView(binding.root)
            .setPositiveButton("Create", null) // We'll set the click listener in onStart()
            .setNegativeButton("Cancel", null)
            .create()

        return dialog
    }

    override fun onStart() {
        super.onStart()

        val dialog = dialog as AlertDialog

        // Override the positive button click listener
        dialog.getButton(AlertDialog.BUTTON_POSITIVE).setOnClickListener {
            val name = binding.editTextDocumentName.text.toString().trim()
            val type = binding.editTextDocumentType.text.toString().trim()
            val topic = binding.editTextDocumentTopic.text.toString().trim()
            val purpose = binding.editTextDocumentPurpose.text.toString().trim()

            // Get selected folder
            val selectedPosition = binding.spinnerFolder.selectedItemPosition
            val selectedFolderId = if (selectedPosition > 0) {
                folderIdList[selectedPosition]
            } else {
                null // "No Folder" selected
            }
```

```
            if (name.isNotEmpty()) {
                // Show loading indicator
                showLoadingIndicator()

                // Call createDocument() from ViewModel
                viewModel.createDocument(name, type, topic, purpose, selectedFolderId)
            } else {
                Toast.makeText(requireContext(), "Document Name cannot be empty", Toast.LENGTH_SHORT).show()
            }
        }

        // Override the negative button click listener
        dialog.getButton(AlertDialog.BUTTON_NEGATIVE).setOnClickListener {
            dialog.dismiss()
        }

        // Now that the view is created, set up the observers
        setupObservers()
    }

    private fun setupObservers() {
        // Use 'this' as the LifecycleOwner
        viewModel.createdDocument.observe(this, Observer { document ->
            if (document != null) {
                // Hide loading indicator
                hideLoadingIndicator()

                // Dismiss the dialog
                dismiss()

                // Open EditorActivity with the new document
                val intent = EditorActivity.newIntent(requireContext(), document)
                startActivity(intent)

                // Reset the LiveData to avoid repeated navigation
                viewModel.resetCreatedDocument()
            }
        })

        // Observe folders and update the Spinner
        viewModel.allFolders.observe(this, Observer { folders ->
            updateFolderSpinner(folders)
        })

        // Initialize the Spinner
        setupFolderSpinner()
    }

    private fun setupFolderSpinner() {
        // Clear any existing data
        folderNameList.clear()
        folderIdList.clear()
```

```
        // Add a default option for "No Folder"
        folderNameList.add("No Folder")
        folderIdList.add(null)

        // Initialize the adapter
        folderAdapter = ArrayAdapter(requireContext(), android.R.layout.simple_spinner_item, folderNameList)
        folderAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
        binding.spinnerFolder.adapter = folderAdapter
    }

    private fun updateFolderSpinner(folders: List<Folder>) {
        // Remove previous folder entries but keep the "No Folder" option at index 0
        if (folderNameList.size > 1) {
            folderNameList.subList(1, folderNameList.size).clear()
            folderIdList.subList(1, folderIdList.size).clear()
        }

        // Add folders to the lists
        for (folder in folders) {
            folderNameList.add(folder.name)
            folderIdList.add(folder.id)
        }

        // Notify the adapter that the data has changed
        folderAdapter.notifyDataSetChanged()
    }

    private fun showLoadingIndicator() {
        binding.progressBar.visibility = View.VISIBLE
    }

    private fun hideLoadingIndicator() {
        binding.progressBar.visibility = View.GONE
    }

    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }
}
```

DocumentAdapter.kt

```
class DocumentAdapter(
    private val onItemClick: (Document) -> Unit
) : ListAdapter<Document , DocumentAdapter.DocumentViewHolder>(DocumentDiffCallback()) {

    override fun onCreateViewHolder(parent: ViewGroup , viewType: Int): DocumentViewHolder {
        val view = LayoutInflater.from(parent.context)
            .inflate(R.layout.item_document, parent, false)
        return DocumentViewHolder(view)
    }
```

```
    override fun onBindViewHolder(holder: DocumentViewHolder, position: Int) {
        val document = getItem(position)
        holder.bind(document)
    }

    inner class DocumentViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        private val textViewDocumentName: TextView = itemView.findViewById(R.id.textViewDocumentName)
        private val textViewDocumentType: TextView = itemView.findViewById(R.id.textViewDocumentType)
        private val textViewDocumentDate: TextView = itemView.findViewById(R.id.textViewDocumentDate)

        fun bind(document: Document) {
            textViewDocumentName.text = document.name
            textViewDocumentType.text = document.type
            textViewDocumentDate.text = formatDate(document.dateCreated)

            itemView.setOnClickListener {
                onItemClick(document)
            }


        }

        private fun formatDate(timestamp: Long): String {
            val sdf = java.text.SimpleDateFormat("MMM dd, yyyy", java.util.Locale.getDefault())
            return sdf.format(java.util.Date(timestamp))
        }
    }

    class DocumentDiffCallback : DiffUtil.ItemCallback<Document>() {
        override fun areItemsTheSame(oldItem: Document, newItem: Document): Boolean {
            return oldItem.id == newItem.id
        }

        override fun areContentsTheSame(oldItem: Document, newItem: Document): Boolean {
            return oldItem == newItem
        }
    }
}
```

DocumentDao.kt

```
@Dao
interface DocumentDao {
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertDocument(document: Document)

    @Query("SELECT * FROM Document WHERE deleted = 0")
    fun getAllDocuments(): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE deleted = 1")
    fun getDeletedDocuments(): LiveData<List<Document>>
```

```
    @Query("SELECT * FROM Document WHERE deleted = 0 ORDER BY type ASC")
    fun getDocumentsSortedByType(): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE deleted = 0 ORDER BY topic ASC")
    fun getDocumentsSortedByTopic(): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE deleted = 0 ORDER BY purpose ASC")
    fun getDocumentsSortedByPurpose(): LiveData<List<Document>>

    @Query("UPDATE Document SET folderId = NULL WHERE folderId = :folderId")
    suspend fun removeFolderFromDocuments(folderId: String)

    @Query("SELECT * FROM Document WHERE deleted = 0 ORDER BY name ASC")
    fun getDocumentsSortedByName(): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE deleted = 0 ORDER BY dateCreated DESC")
    fun getDocumentsSortedByDate(): LiveData<List<Document>>

    // Add other sorting methods...

    @Query("SELECT * FROM Document WHERE deleted = 0 AND name LIKE :query ORDER BY name ASC")
    fun searchDocumentsSortedByName(query: String): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE deleted = 0 AND name LIKE :query ORDER BY dateCreated DESC")
    fun searchDocumentsSortedByDate(query: String): LiveData<List<Document>>

    // Add other search and sort methods...

    @Query("SELECT * FROM Document WHERE deleted = 0 AND name LIKE :query")
    fun searchDocuments(query: String): LiveData<List<Document>>

    @Query("SELECT * FROM Document WHERE folderId = :folderId AND deleted = 0 ORDER BY name ASC")
    fun getDocumentsInFolder(folderId: String): LiveData<List<Document>>

    @Update
    suspend fun updateDocument(document: Document)

    @Query("SELECT * FROM Document WHERE id = :documentId")
    fun getDocumentById(documentId: String): LiveData<Document>

}
```

ДОДАТОК Г

Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА МЕТОДУ ТА ЗАСОБІВ СИСТЕМИ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ І РЕДАГУВАННЯ ТЕКСТОВИХ ДОКУМЕНТІВ

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Магістерська кваліфікаційна робота на тему:
«Розробка методу та засобів системи автоматизованого формування і редагування текстових документів»

Автор: ст.групи 2ПІ-23м Музичук Д.Р.
Науковий керівник: к.т.н., доцент каф. ПЗ Войтко В.В.

Вінниця - 2024

Рисунок Г.1 – Титульний слайд

Актуальність теми

Робота з документами є невід'ємною частиною будь-якого робочого процесу в різних галузях, починаючи від бізнесу та закінчуючи науковою діяльністю.

Значний поштовх у розвиток документообігу задає сучасна інформаційна епоха, яка генерує велику кількість знань та інформації, яку необхідно відповідним чином оформити та впорядкувати для ефективного її використання. Також, Інтернет значно полегшив процес обміну документами та дозволив користувачам на великій відстані один від одного спільно працювати над створенням різного роду документації.

Документи використовуються для збереження, передачі та представлення інформації, тому важливо, щоб процес їх створення був максимально ефективним, точним і зручним для користувача. З огляду на швидкий розвиток технологій і необхідність обробляти великі обсяги інформації, автоматизація роботи з документами стає дедалі актуальнішою. Впровадження інструментів для автоматичного створення та редагування документів дозволяє зекономити час, уникнути помилок і зробити документи більш професійними та структурованими. Тому актуальною є розробка системи автоматизованого формування і редагування текстових документів.

2

Рисунок Г.2 – Актуальність теми

Мета, об’єкт та предмет дослідження

Метою магістерської кваліфікаційної роботи є підвищення рівня автоматизації процесу розробки текстових документів шляхом розробки та впровадження системи автоматизованого формування і редагування документів, що дозволить підвищити якість та надійність обробки текстових файлів.

Об’єктом дослідження є процес розробки системи для редагування та формування текстових документів.

Предметом дослідження є методи та засоби розробки системи для редагування та формування текстових документів.

3

Рисунок Г.3 – Мета, об’єкт та предмет дослідження

Завдання дослідження

– розробити модель, методи та алгоритми роботи системи для автоматичного формування і редагування текстових документів;

– розробити функціонал для автоматизованого структурування документів;

– розробити функціонал для інтерактивних порад щодо наповнення документів вмістом;

– розробити функціонал для порад щодо наповнення документів релевантним додатковим вмістом (наприклад таблиці, зображення, графіки);

– налаштувати застосунок для роботи на різних видах пристроїв;

– провести тестування розробленої системи.

4

Рисунок Г.4 – Завдання дослідження

Наукова новизна

1. Подальшого розвитку отримав метод формування структури документа, який, на відміну від відомих, використовує штучний інтелект для формування розділів та структурних частин документа, враховуючи його тему та призначення, що дозволить автоматизувати процес редагування документів та підвищити надійність процесу формування текстової документації.
2. Подальшого розвитку отримав метод рекомендацій наповнення документа, який, на відміну від відомих, використовує штучний інтелект для надання точних рекомендацій залежно від теми та призначення документа, що дозволить підвищити ефективність створення текстових документів.
3. Подальшого розвитку отримав метод рекомендацій елементів візуалізації, який, на відміну від відомих, відстежує зміни документа та додає елементи візуалізації для підвищення інформативності документа та згідно з його контекстом, що дозволить покращити інтерфейсні можливості системи в процесі автоматизованого формування текстових документів.
4. Подальшого розвитку отримав метод роботи з елементами візуалізації у Android View інтерфейсі текстового редактора, який, на відміну від існуючих, використовує діалогові вікна та конвертацію елементів візуалізації у зображення, що дозволить покращити інтерфейсні можливості системи у процесі взаємодії з користувачем.

5

Рисунок Г.5 – Наукова новизна

Практична цінність отриманих результатів

Розроблена система може використовуватися для прискорення та полегшення процесу формування й редагування текстових документів.

6

Рисунок Г.6 – Практична цінність отриманих результатів

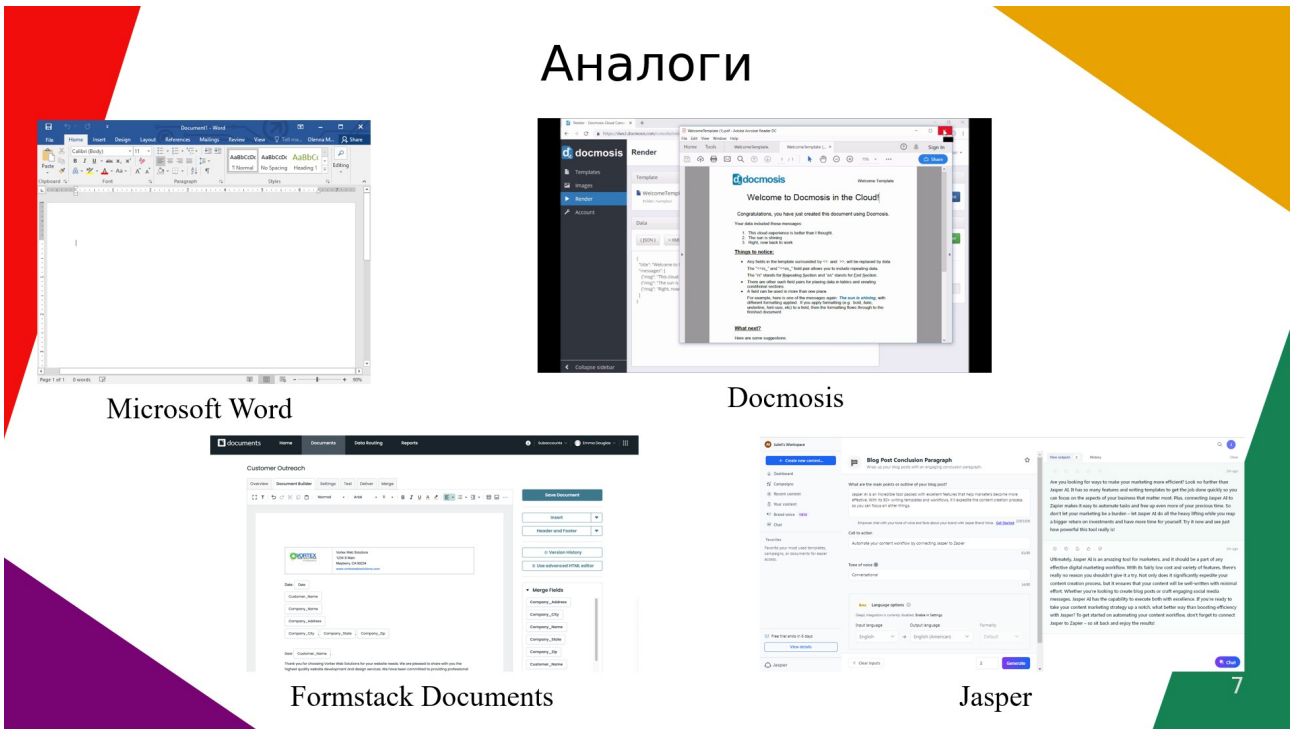

Рисунок Г.7 – Аналоги

Порівняльний аналіз аналогів

	Microsoft Word	Formstack Documents	Docmosis	Jasper	Власна розробка
Перевірка граматики	1	1	0	1	1
Текстовий редактор	1	0	1	1	1
Шаблони для документів	1	0	1	0	1
Динамічні рекомендації елементів візуалізації у документі	1	0	0	0	1
Поради щодо текстового наповнення документа	1	1	0	1	1
Динамічне структурування документа	0	0	1	0	1
Загальна оцінка	4	2	3	3	6

8

Рисунок Г.8 – Порівняльний аналіз аналогів

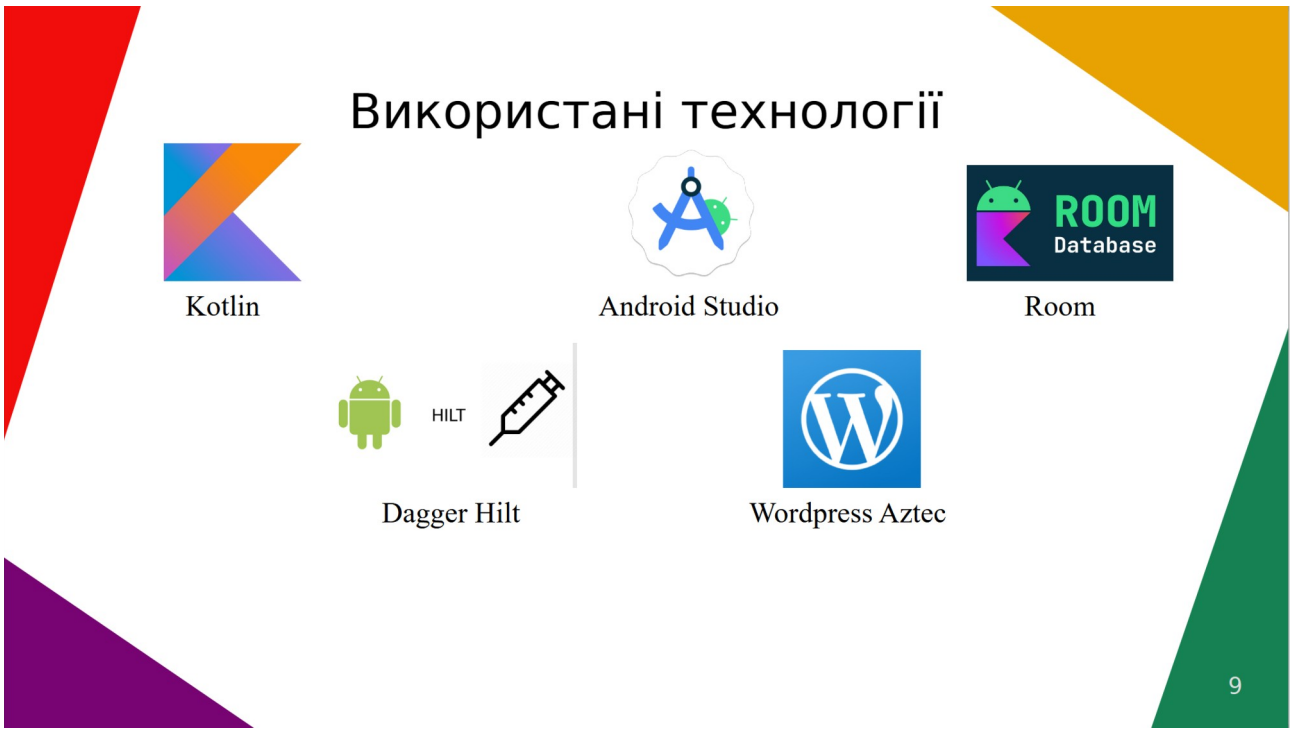

Рисунок Г.9 – Використані технології

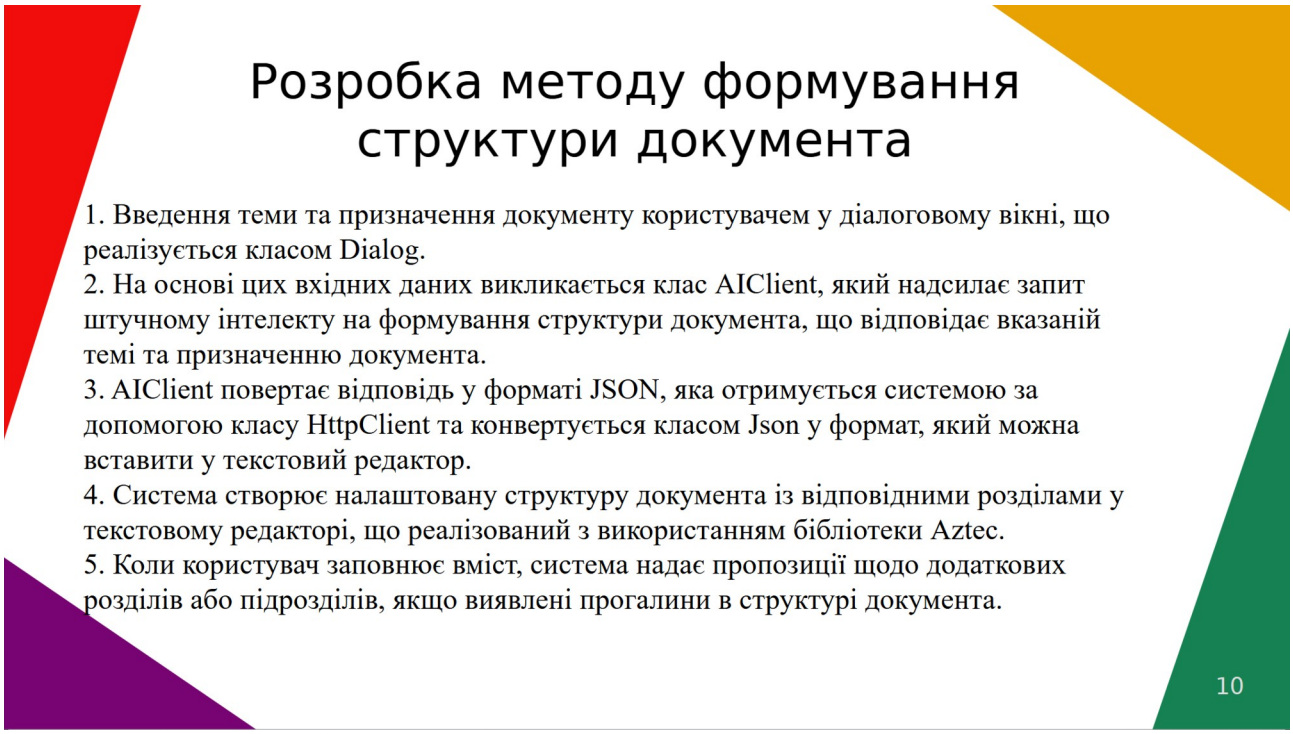

Рисунок Г.10 – Розробка методу формування структури документа

Рисунок Г.11 – Блок-схема алгоритму роботи методу формування структури документа

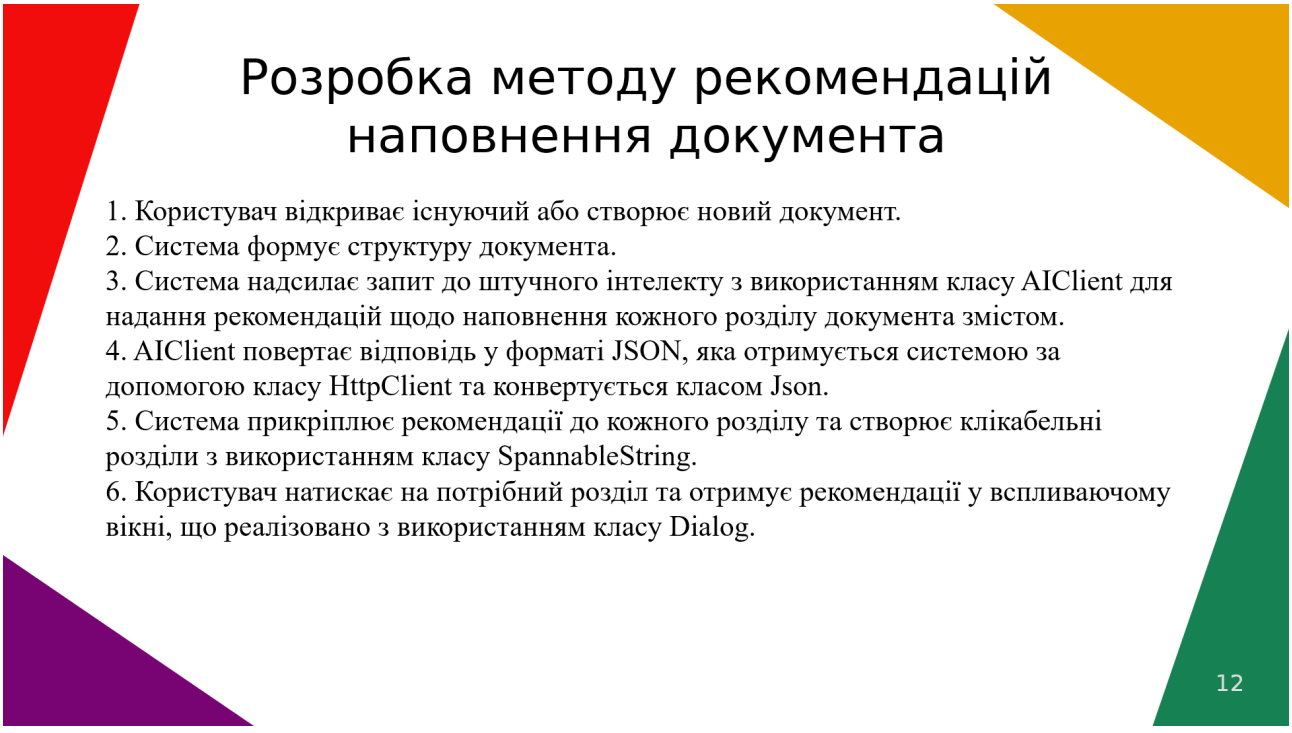

Рисунок Г.12 – Розробка методу рекомендацій наповнення документа

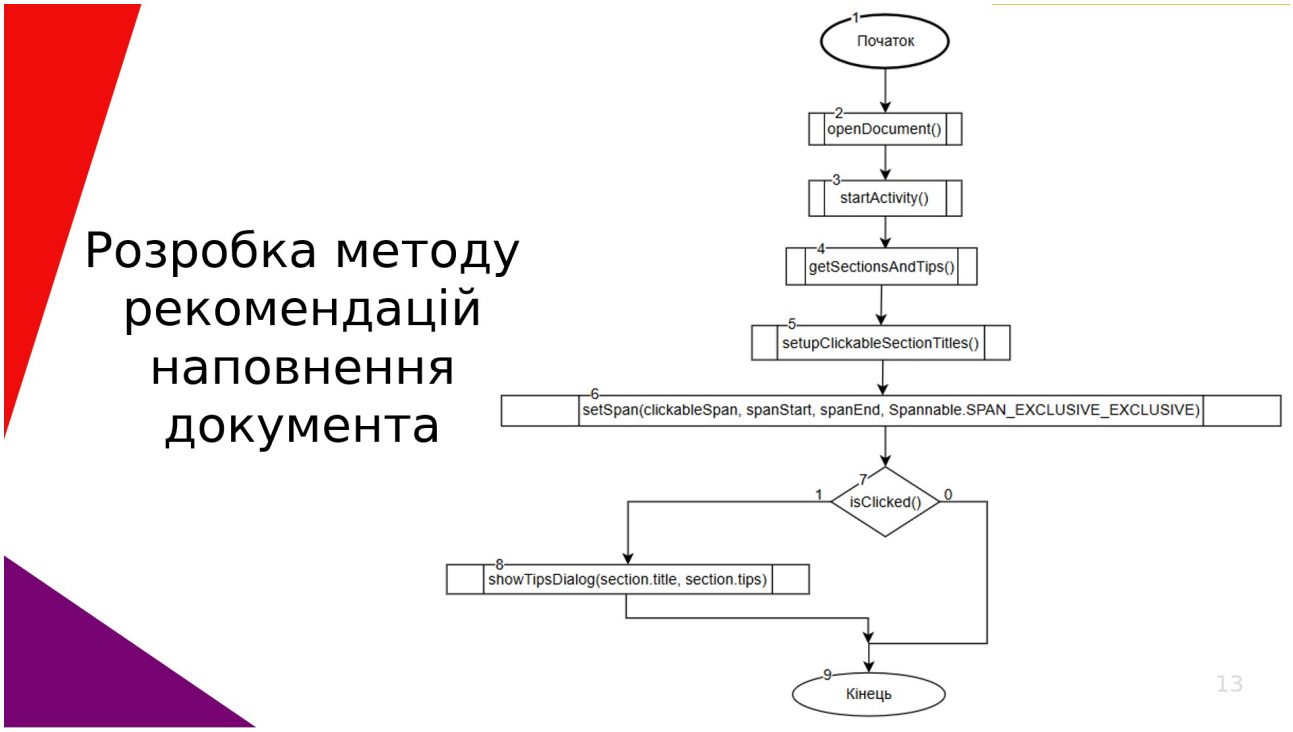

Рисунок Г.13 – Блок-схема алгоритму роботи методу рекомендацій наповнення документа

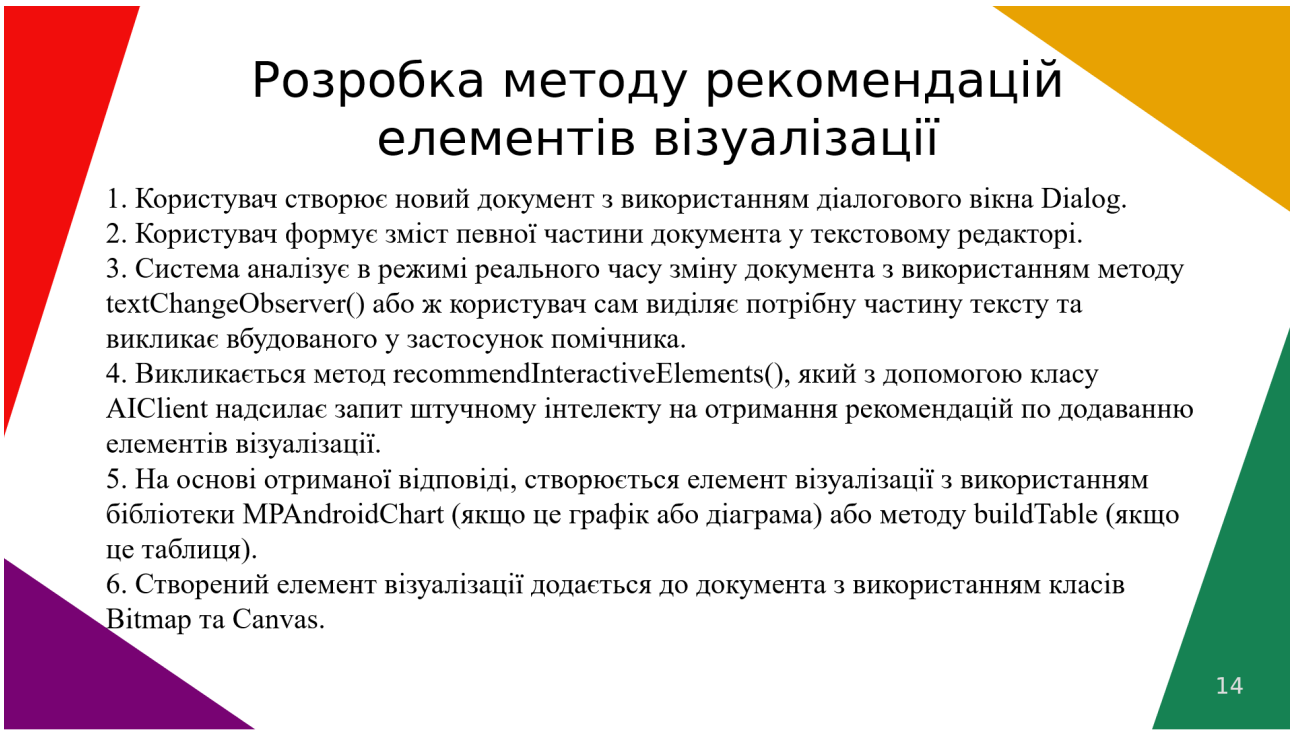

Рисунок Г.14 – Розробка методу рекомендацій елементів візуалізації

Рисунок Г.15 – Блок-схема алгоритму роботи методу рекомендацій елементів візуалізації

Рисунок Г.16 – Розробка методу роботи з елементами візуалізації у Android View інтерфейсі текстового редактора

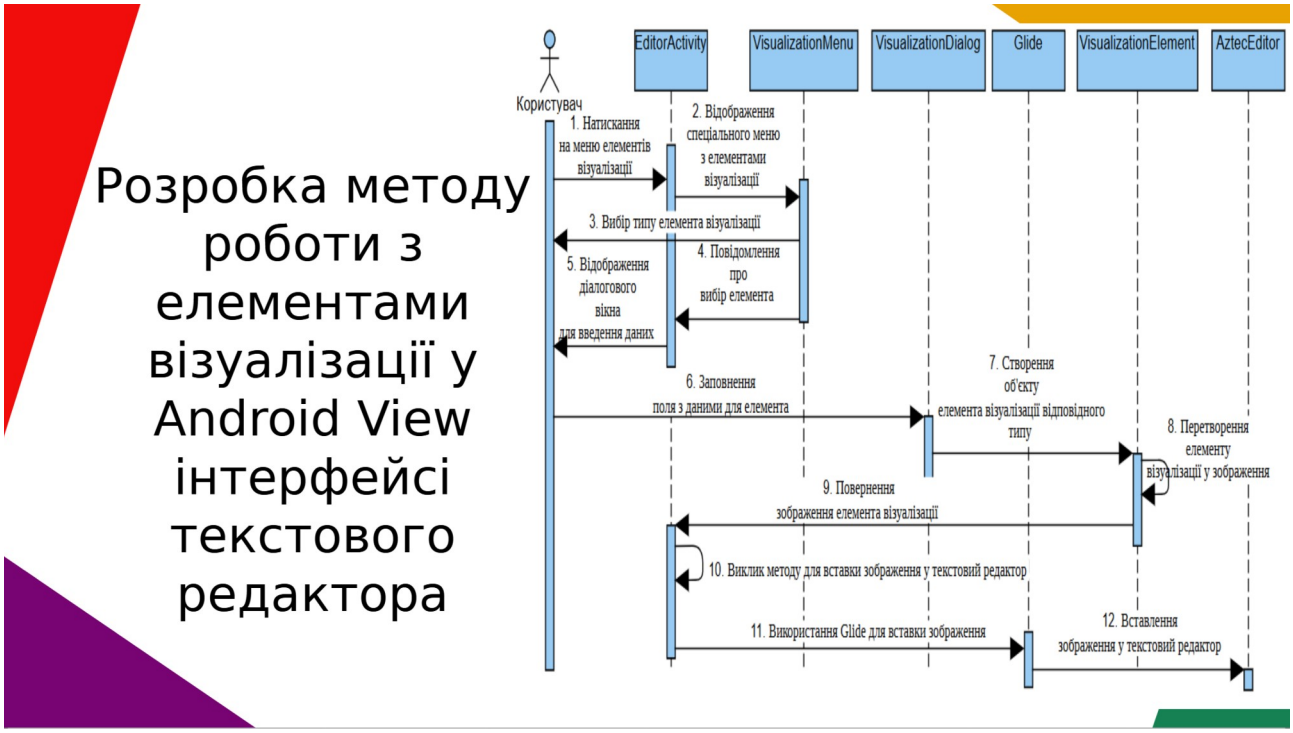

Рисунок Г.17 – Діаграма послідовності методу роботи з елементами візуалізації у Android View інтерфейсі текстового редактора

Рисунок Г.18 – Модель системи

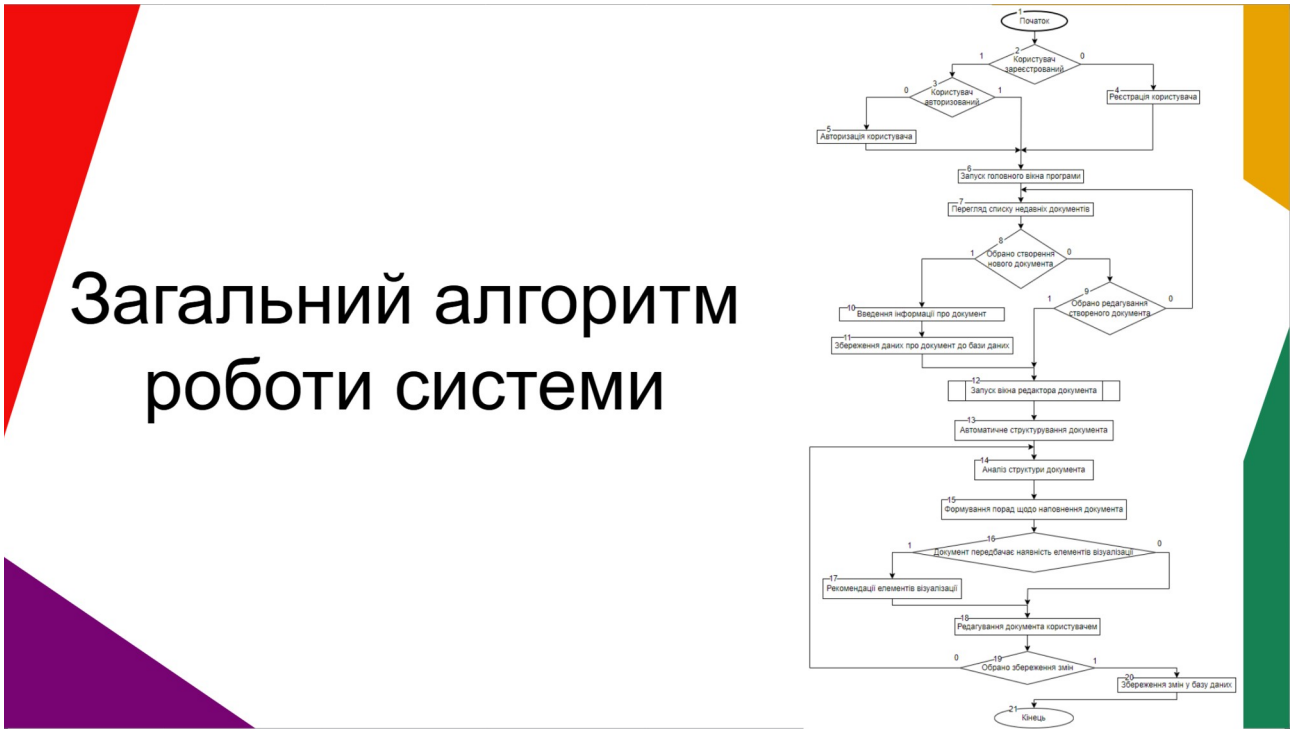

Рисунок Г.19 – Загальний алгоритм роботи системи

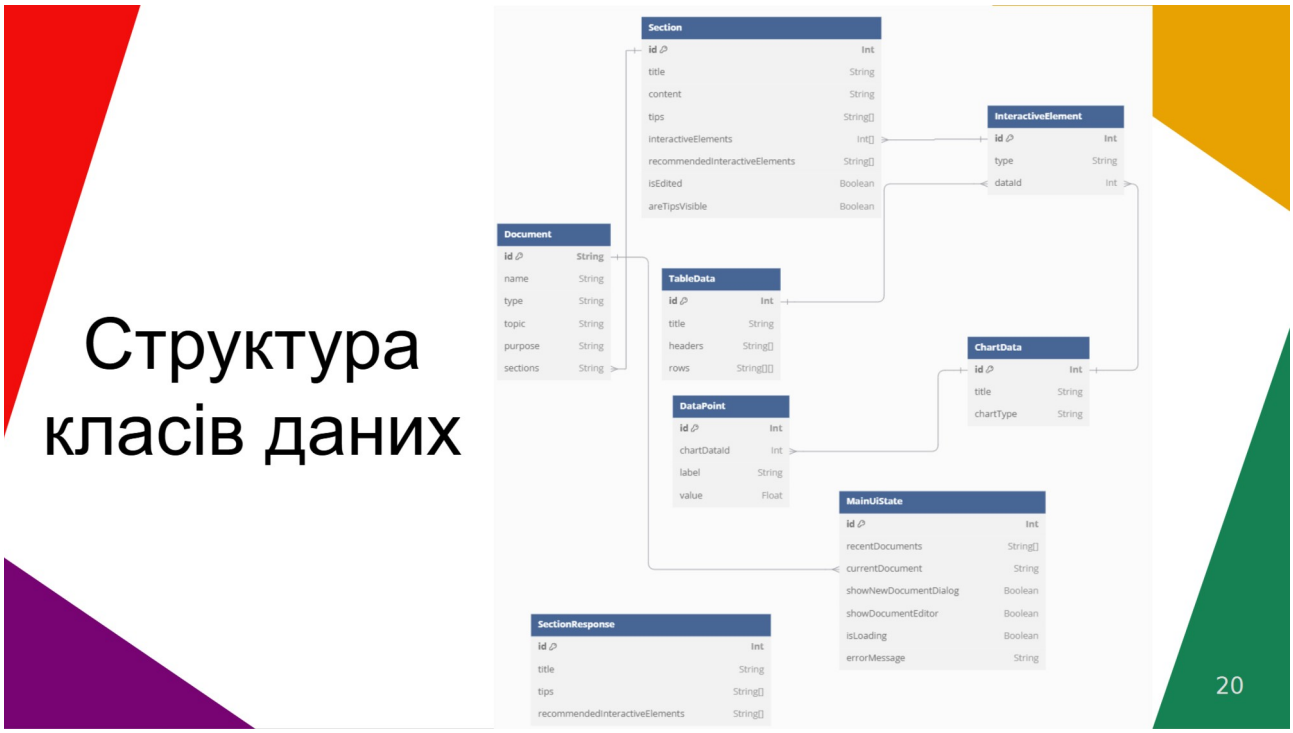

Рисунок Г.20 – Структура класів даних

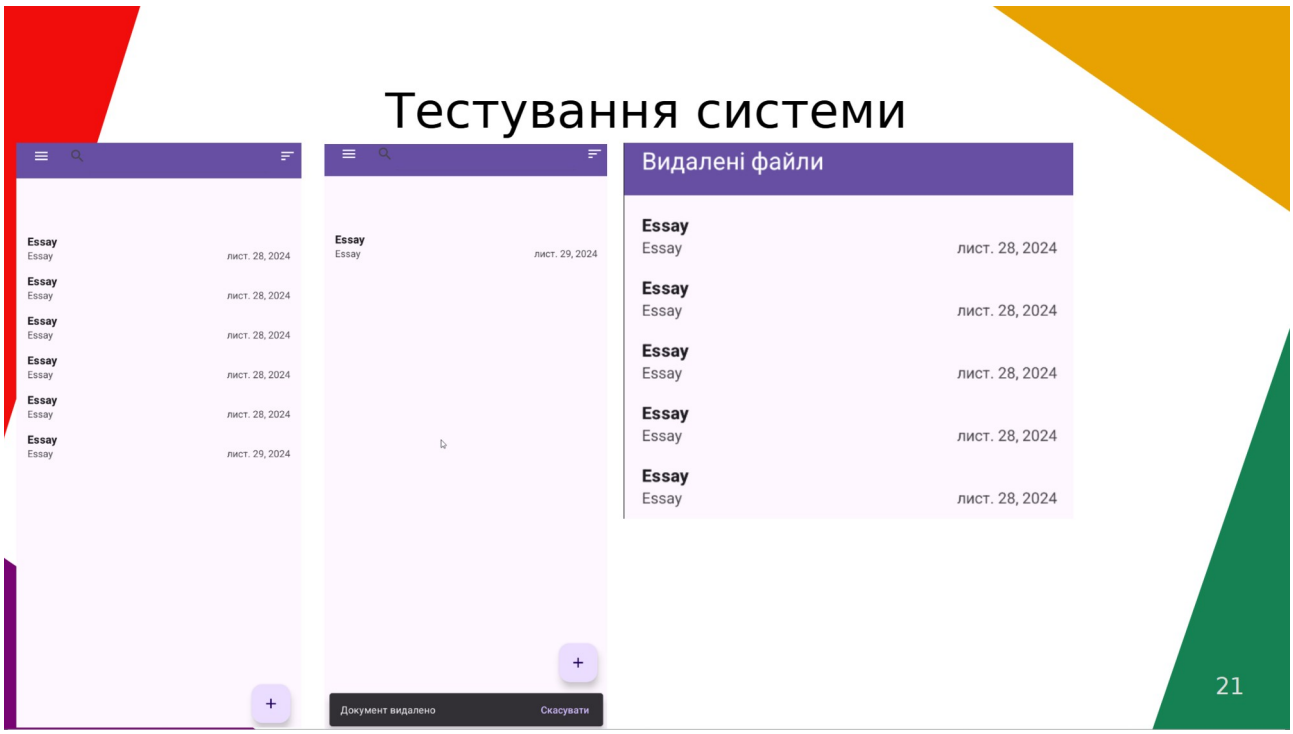

Рисунок Г.21 – Тестування системи (частина 1)

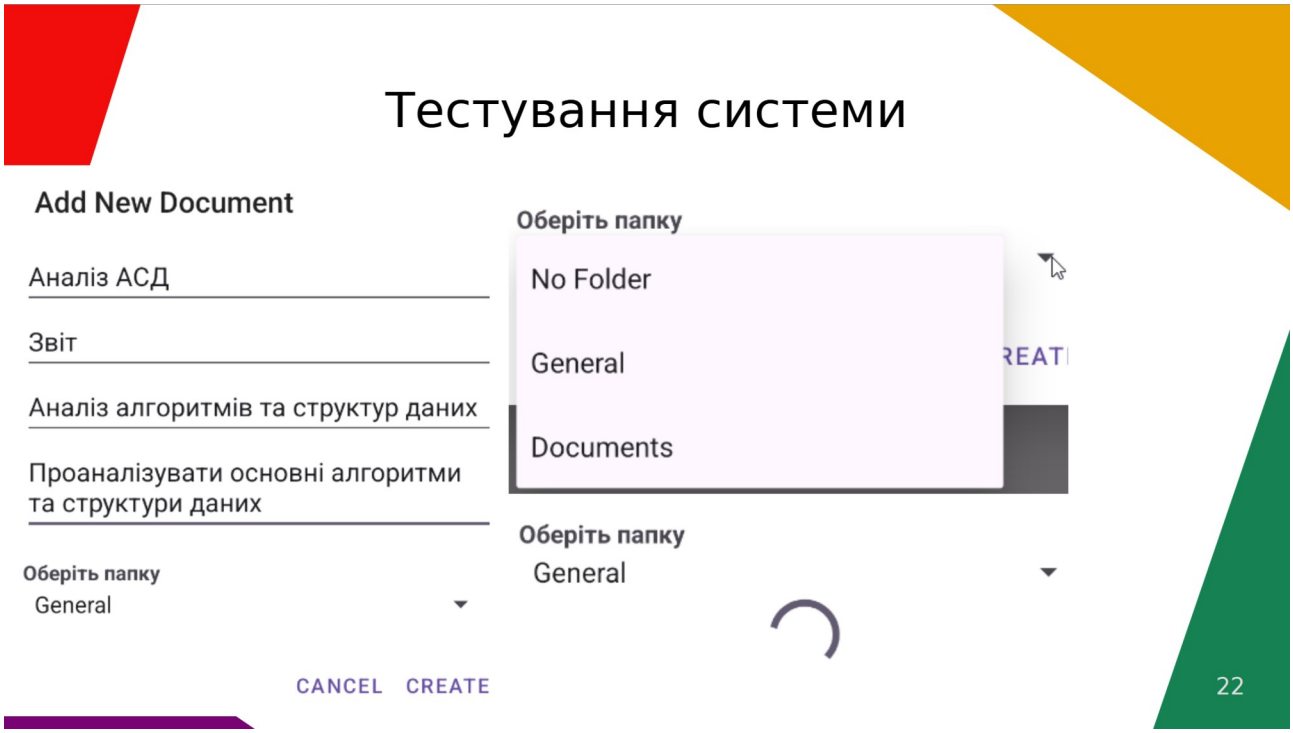

Рисунок Г.22 – Тестування системи (частина 2)

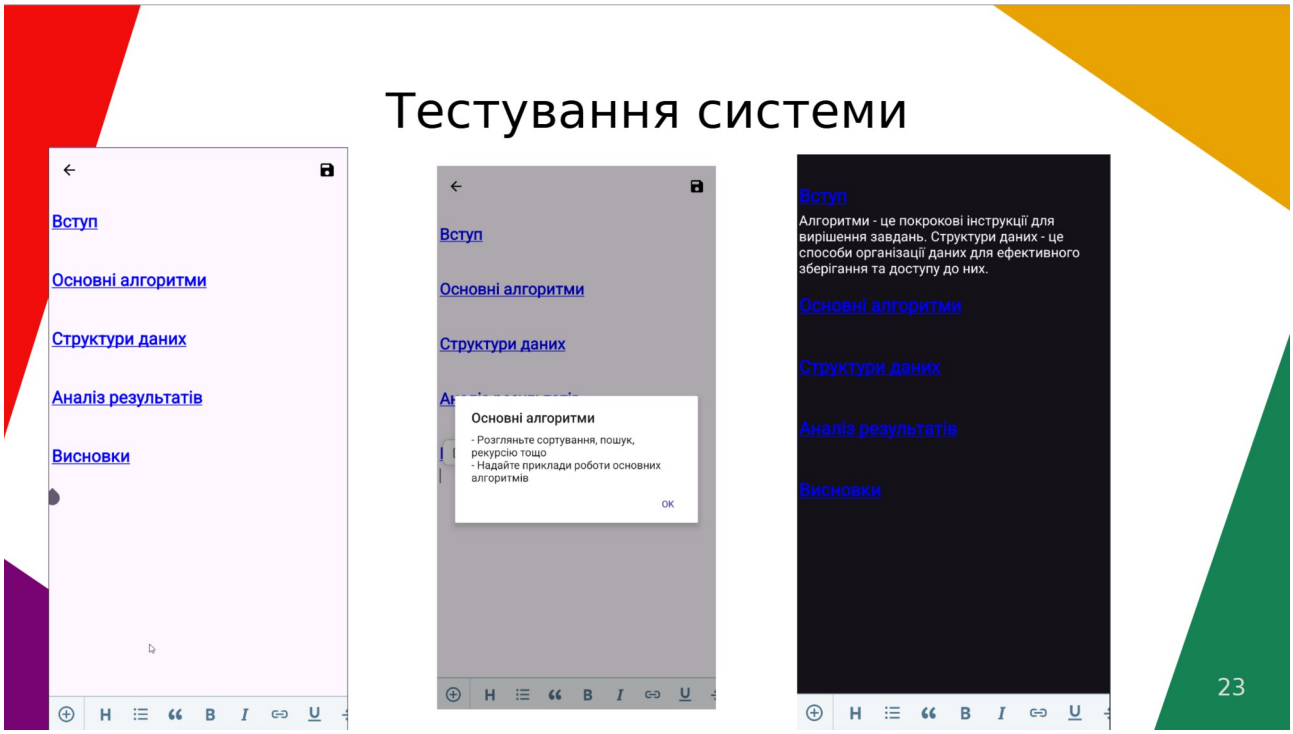

Рисунок Г.23 – Тестування системи (частина 3)

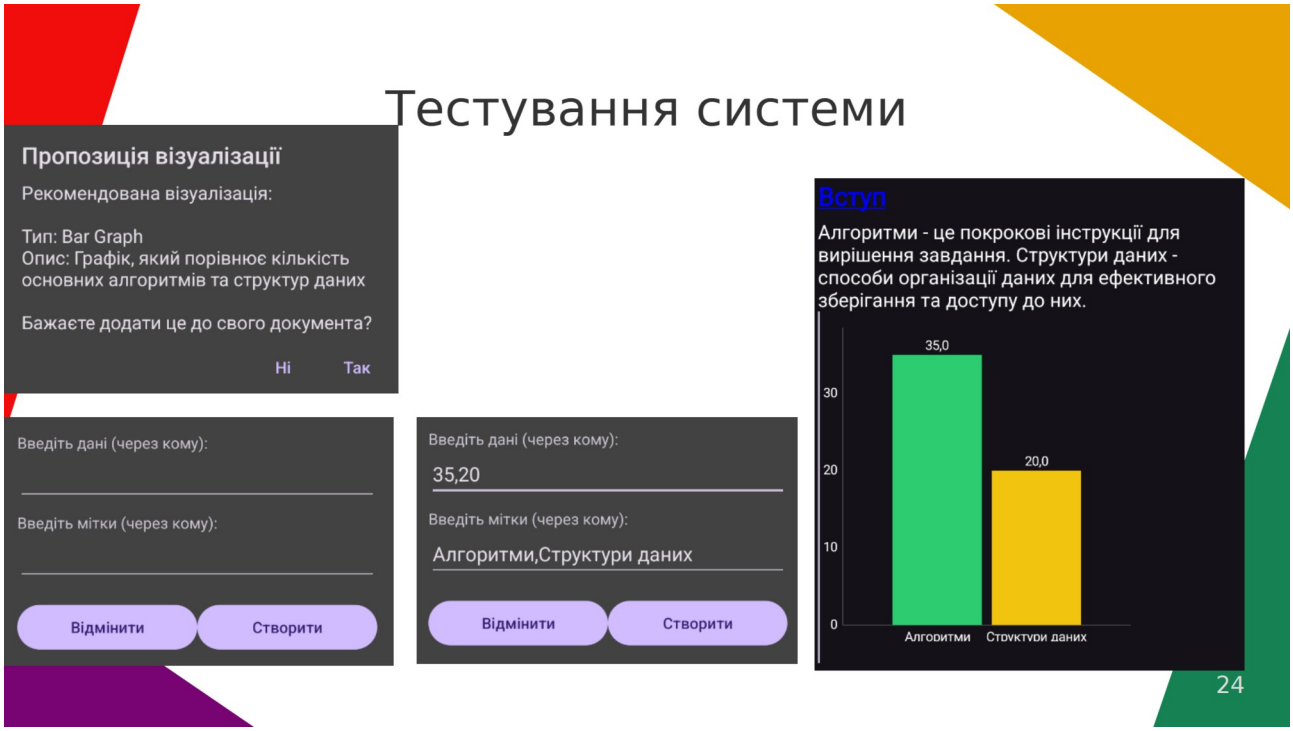

Рисунок Г.24 – Тестування системи (частина 4)

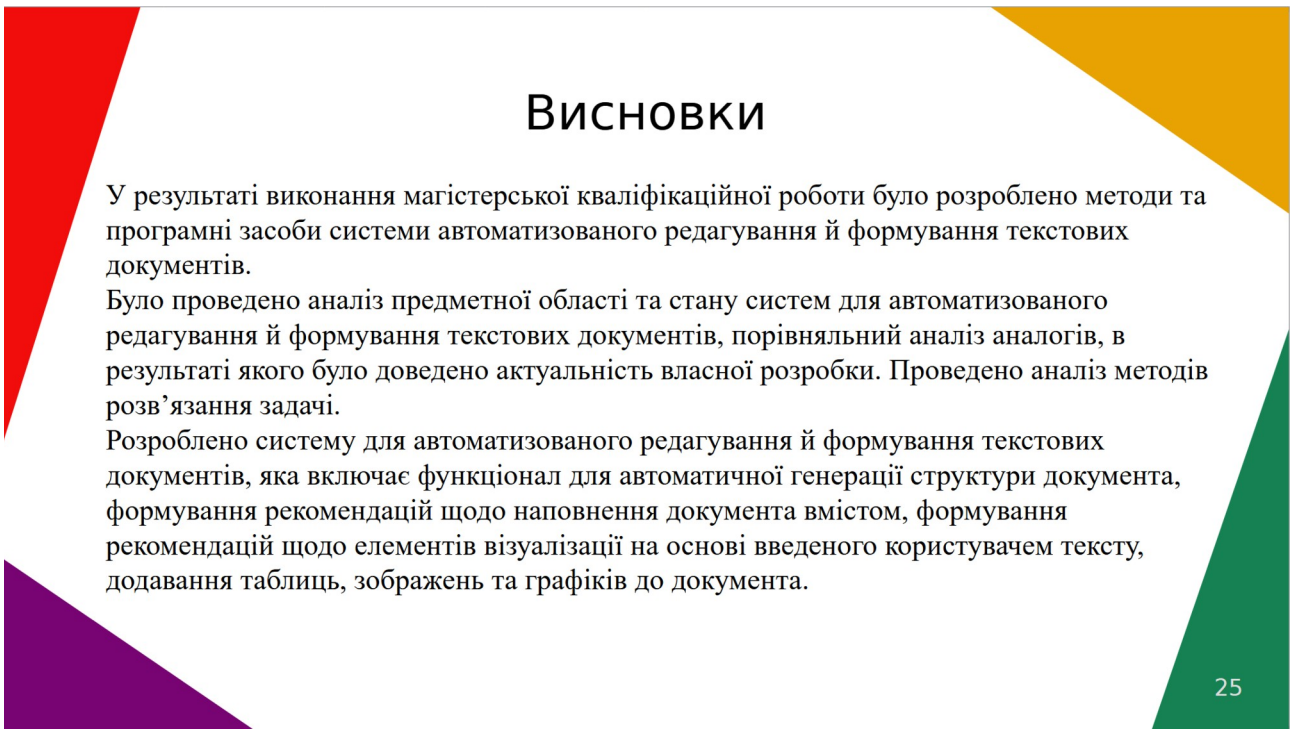

Рисунок Г.25 – Висновки (частина 1)

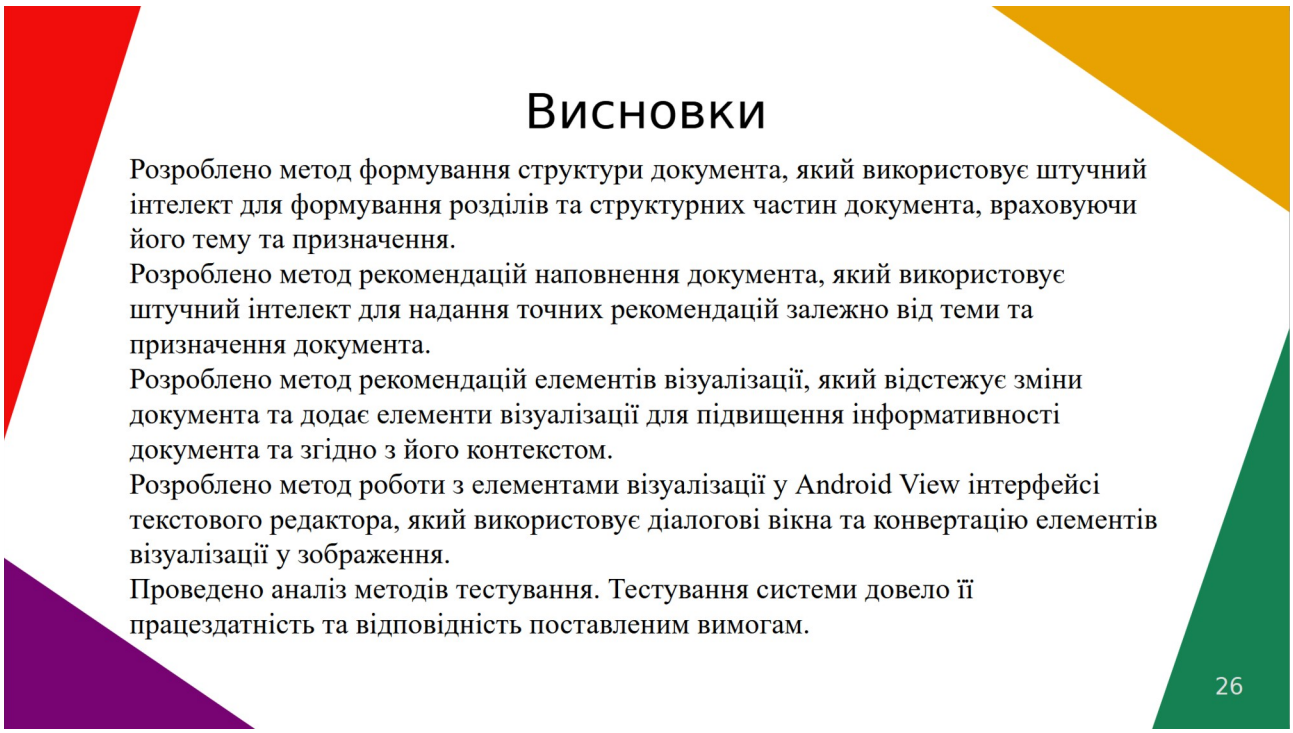

Рисунок Г.26 – Висновки (частина 2)

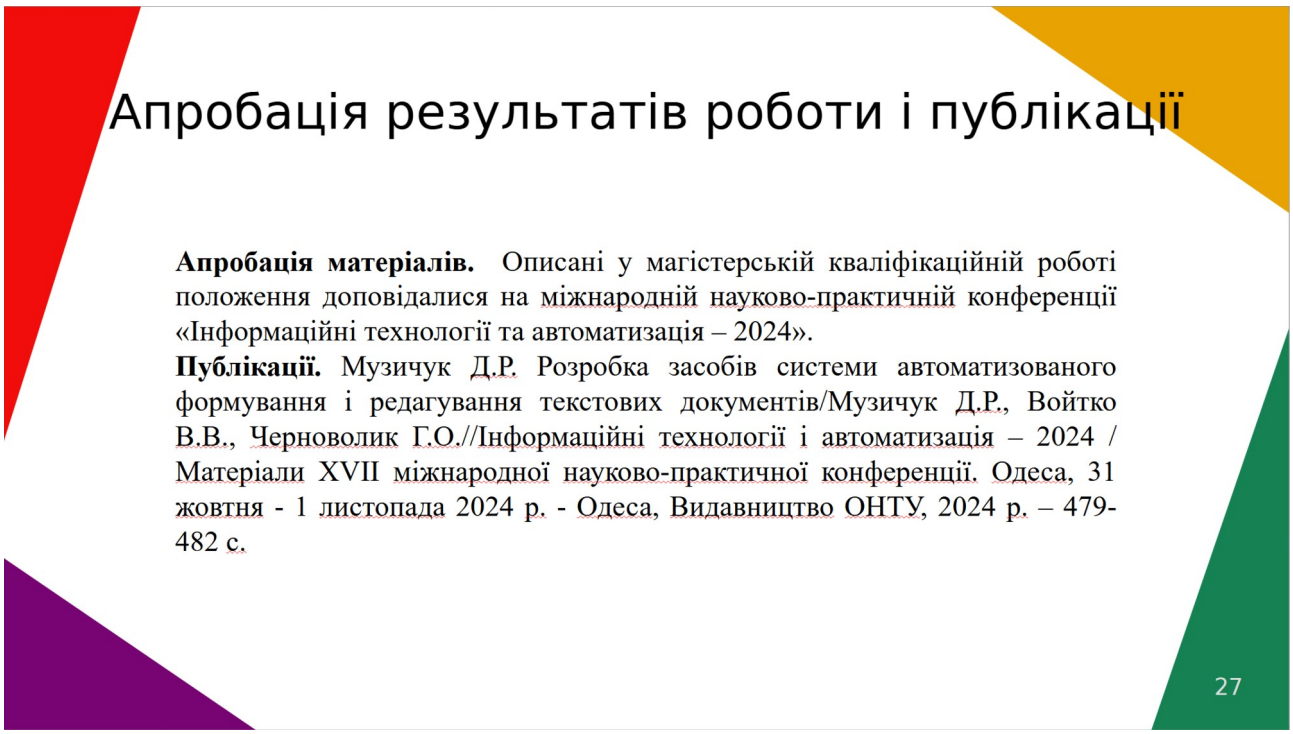

Рисунок Г.27 – Апробація результатів роботи і публікації

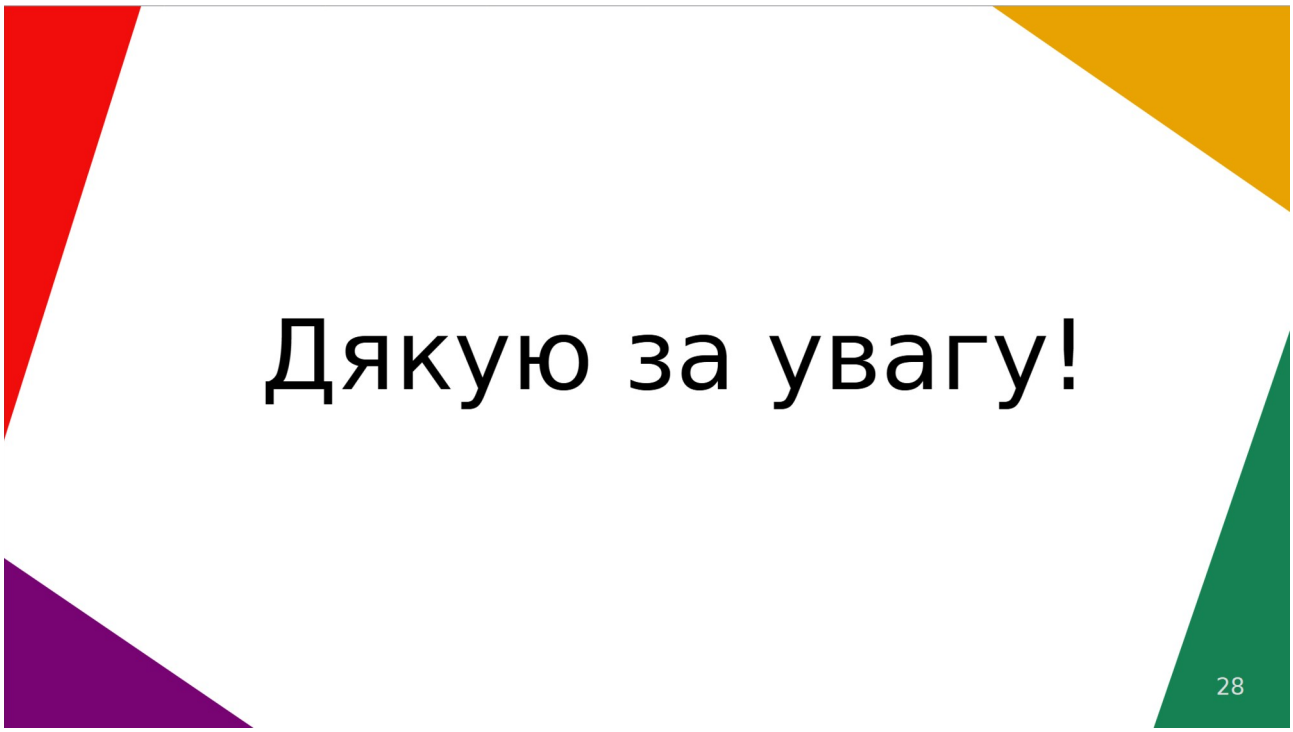

Рисунок Г.28 – Фінальний слайд