Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка програмного забезпечення для автоматизованої генерації технічної документації з використанням фреймворку LangChain

Виконав: студент IV курсу, групи СП-41
спеціальності 121 – Інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Бойко В.Р. (прізвище та ініціали)

Керівник (підпис) Стоянов Ю.М. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю.М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М.Р. (прізвище та ініціали)

Рецензент (підпис) Ревнюк О.А. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ проф. Петрик М.Р.
(підпис) (прізвище та ініціали)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 Інженерія програмного забезпечення
(шифр і назва спеціальності)

студенту Бойку Володимиру Руслановичу

1. Тема роботи Розробка програмного забезпечення для автоматизованої генерації технічної документації з використанням фреймворку LangChain

Керівник роботи Стоянов Юрій Миколайович к.т.н., доц. каф. ПІ
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом по університету від « 06 » квітня 2026 року № _____

2. Термін подання студентом роботи 22.06.2026

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз вимог та огляд предметної області.
2. Проєктування та конструювання
3. Тестування.
4. Безпека життєдіяльності, основи охорони праці.
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку..
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 6 квітня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	*Виконано*
2.	*Робота над першим розділом «Огляд придметної області»*	*13.04 – 26.04*	*Виконано*
3.	*Робота над другим розділом «Проєктування та реалізація»*	*27.04 – 03.05*	*Виконано*
4.	*Робота над третім розділом «Тестування»*	*04.05 – 17.05*	*Виконано*
	Робота над четвертим розділом «Безпека життєдіяльності, основи охорони праці»	*18.05 – 24.05*	*Виконано*
5.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	*Виконано*
6.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	*Виконано*
7.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	*Виконано*
8.	*Захист кваліфікаційної роботи бакалавра*		
9.			
10.			
11.			
12.			

Студент _____ Бойко В.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Стоянов Ю.М.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бойко В. Р. Розробка десктопного програмного забезпечення для автоматизованої інтелектуальної генерації багатоформатної технічної документації на основі статичного аналізу вихідного коду проєкту з використанням фреймворку LangChain та великих мовних моделей : робота на здобуття кваліфікаційного ступеня бакалавра : спец. 121 - інженерія програмного забезпечення / наук. кер. Ю. М. Стоянов. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2026. 70 с. // С. - 54, табл. - 0, рис. - 10, бібліогр. – 26, слайд. –18, додат. - 2 ,

Ключові слова: генерація документації, статичний аналіз, AST, великі мовні моделі, LangChain, автоматизація розробки.

Мета роботи – створення десктопного програмного забезпечення для інтелектуальної генерації технічної документації на основі аналізу коду з використанням LangChain та мовних моделей.

У першому розділі здійснено огляд предметної області та обґрунтовано доцільність використання великих мовних моделей замість класичних генераторів.

У другому розділі спроєктовано архітектуру системи, реалізовано модуль статичного аналізу (AST), конвеєр LangChain та багатопотоковий інтерфейс на базі PyQt6.

У третьому розділі проведено функціональне тестування застосунку та перевірено компонент асинхронного обміну даними.

У четвертому розділі розглянуто класифікацію небезпечних виробничих факторів та вплив вібрації.

Об’єктом дослідження є процес створення веб-сервісу для комерційного розповсюдження аудіопродукції з механізмами захисту авторських прав. Предметом дослідження є методи статичного аналізу коду, інженерія підказок та

технології побудови конвеєрів (LangChain). Методи дослідження: системний аналіз, парсинг через AST, об'єктно-орієнтоване проєктування та функціональне тестування.

ABSTRACT

Boyko V. R. Development of desktop software for automated intelligent generation of multi-format technical documentation based on static analysis of the project's source code using the LangChain framework and large language models : bachelor's thesis : spec. 121 - Software Engineering / sci. adv. Yu. M. Stoianov. Ternopil : Ivan Puluj Ternopil National Technical University, 2026. 70 p. // P. - 54, tab. - 0, fig. - 10, ref. - 26, slides - 18, app. - 2.

Keywords: documentation generation, static analysis, AST, large language models, LangChain, development automation.

The purpose of the work is the creation of desktop software for the intelligent generation of technical documentation based on code analysis using LangChain and language models. In the first chapter, an overview of the subject area is provided, and the expediency of using large language models instead of classical generators is justified.

In the second chapter, the system architecture is designed, and the static analysis module (AST), LangChain pipeline, and multi-threaded interface based on PyQt6 are implemented. In the third chapter, functional testing of the application is conducted, and the asynchronous data exchange component is verified.

In the fourth chapter, the classification of hazardous occupational factors and the impact of vibration are considered.

The object of research is the process of automated creation of technical documentation for source code using artificial intelligence.

The subject of research is methods of static code analysis, prompt engineering, and pipeline building technologies (LangChain). Research methods: system analysis, parsing via AST, object-oriented design, and functional testing.

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення.

ШІ – штучний інтелект.

LCEL (LangChain Expression Language) – декларативна мова виразів фреймворку LangChain, призначена для зручного створення конвеєрів обробки даних.

LLM (Large Language Model) – велика мовна модель, нейромережа, натренована на великих обсягах текстових даних.

API (Application Programming Interface) – прикладний програмний інтерфейс, набір функцій та структур для взаємодії між програмами.

AST (Abstract Syntax Tree) – абстрактне синтаксичне дерево, структура даних, що відображає синтаксичну структуру вихідного коду.

GUI (Graphical User Interface) – графічний інтерфейс користувача. HTML (HyperText Markup Language) – стандартизована мова розмітки для створення

LCEL (LangChain Expression Language)– декларативна мова виразів фреймворку LangChain, призначена для зручного створення конвеєрів обробки даних.

ЗМІСТ

ВСТУП

Актуальність теми дослідження. Сучасна індустрія розробки програмного забезпечення характеризується стрімким збільшенням складності систем та високою динамікою внесення змін. У таких умовах ручне написання та підтримка технічної документації в актуальному стані вимагає значних часових ресурсів розробників, що уповільнює процес випуску продукту та підвищує ризик її застарівання.

Поява великих мовних моделей відкрила нові можливості для автоматизації інженерних процесів. Проте їх пряме застосування обмежене лімітами контекстного вікна та складністю структурування зв'язків між модулями системи. Для подолання цих обмежень ефективним є використання оркестраційного фреймворку LangChain, який дозволяє створювати гнучкі конвеєри обробки даних, інтегрувати статичний аналіз коду зі структурованими шаблонами підказок та реалізовувати логіку логічного виведення. Таким чином, розробка системи інтелектуальної генерації технічної документації на основі поєднання методів статичного аналізу вихідного коду та можливостей LangChain є актуальним науково-практичним завданням.

Мета дослідження — розробка десктопного програмного забезпечення для автоматизованої інтелектуальної генерації багатоформатної технічної документації на основі статичного аналізу вихідного коду проєкту з використанням фреймворку LangChain та великих мовних моделей.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ

Ця робота заснована на сучасних дослідженнях у галузі програмної інженерії, методів штучного інтелекту та технологій обробки природної мови. Зокрема, в основі дослідження лежить концепція інтеграції інструментів статичного аналізу кодової бази із генеративними можливостями великих мовних моделей. Використання оркестраційного фреймворку LangChain дозволяє підняти процес документування на новий рівень, забезпечуючи побудову гнучких, багатокрокових конвеєрів обробки інформації. Такий підхід базується на засадах автоматизації рутинних процесів життєвого циклу розробки програмного забезпечення і спрямований на підвищення ефективності роботи інженерних команд та мінімізацію технічного боргу.

Ця тема охоплює кілька важливих аспектів, що лежать на стику системного аналізу ПЗ та інженерії знань. По-перше, це синтаксичний аналіз вихідного тексту програм, який включає побудову абстрактних синтаксичних дерев для точного вилучення структури модулів, класів, функцій та сигнатур методів. По-друге, важливим аспектом є інженерія підказок — розробка та оптимізація контекстних шаблонів для мовних моделей, що гарантує отримання структурованих, точних та стандартизованих технічних текстів. По-третє, дослідження охоплює архітектурне проєктування десктопних систем із підтримкою асинхронності та багатопоточності, що необхідно для забезпечення стабільної роботи інтерфейсу користувача під час тривалих мережевих запитів до API мовних моделей.

1.1 Огляд аналогів

Для автоматизації процесу створення технічної документації та підтримки її в актуальному стані в сучасній програмній інженерії використовується широкий спектр інструментальних засобів. Залежно від базових алгоритмів,

архітектурних підходів та ступеня залучення інтелектуальних методів обробки інформації, існуючі аналоги можна класифікувати на дві основні групи: традиційні синтаксичнігенератори, що виконують статичний аналіз коментарів, та сучасні інтелектуальні хмарні платформи на базі штучного інтелекту.

1.1.1 Традиційні генератори документації

Інструменти сімейства Sphinx та Doxygen є класичними представниками систем автоматизації документування, які використовуються в індустрії протягом десятиліть.

Sphinx — це генератор документації, спочатку розроблений для мови програмування Python, який використовує текстовий формат reStructuredText або Markdown як вхідні дані. Його інтеграція з кодовою базою здійснюється за допомогою розширень (наприклад, autodoc), які виконують статичний та динамічний імпорт модулів, сканують синтаксичну структуру програми та вилучають заздалегідь написані розробником рядки документації (docstrings).

Doxygen функціонує за схожим принципом, але орієнтований на ширший спектр мов програмування (C++, Java, C#, PHP тощо). Він аналізує вихідний код як набір токенів, будує внутрішню структуру зв'язків між класами та методами й вилучає коментарі, що оформлені за спеціальними тегами (наприклад, @param, @return).

1.2 Огляд існуючих технологій реалізації

Розробка програмного забезпечення для автоматизованої генерації технічної документації на основі вихідного коду є комплексною інженерною задачею, реалізація якої вимагає інтеграції декількох різних за своєю природою технологічних стеків. Фундаментом інтелектуального аналізу вихідного тексту програми є великі мовні моделі. Сучасний інженерний підхід пропонує використання комерційних хмарних прикладних інтерфейсів розробника, серед яких оптимальним вибором за балансом швидкісних характеристик, вартості

обробки токенів та якості розуміння логіки програмування є архітектури сімейства.

GPT, зокрема модель gpt-4o-mini. Ці моделі навчалися на колосальних обсягах вихідного коду з публічних репозиторіїв, тому вони здатні точно розпізнавати складні алгоритми, виявляти причинно-наслідкові зв’язки між функціями та формувати технічні описи природною мовою. Інтеграція з ними здійснюється через захищені хмарні запити, де для забезпечення детермінованості технічного тексту встановлюються низькі значення температури генерації та чіткі ліміти максимальної кількості токенів у відповіді.

Для ефективного керування взаємодією з мовними моделями, усунення обмежень низькорівневих викликів прикладних інтерфейсів та динамічного структурування контексту використовується оркестраційний фреймворк LangChain. Його архітектура побудована навколо концепції створення ланцюжків, де вихідні дані одного компонента стають вхідними для іншого. Фреймворк надає гнучкі механізми шаблонів підказок для відокремлення системних інструкцій від динамічних даних коду, а також спеціалізовані парсери виведення, що автоматично очищають та структурують згенерований текст у чистий Markdown. Особливістю сучасних версій цього фреймворку є використання спеціалізованої мови виразів LangChain, яка дозволяє декларативно зв'язувати компоненти через оператор конвеєризації, спрощуючи асинхронне виконання запитів та обробку помилок мережі.

1.3 Висновки до першого розділу

У першому розділі здійснено огляд предметної області та проаналізовано підходи до автоматизованої генерації технічної документації. Встановлено, що класичні синтаксичні генератори мають суттєві функціональні обмеження, оскільки їхня робота базується виключно на статичному вилученні заздалегідь написаних розробником коментарів. Для вирішення цієї проблеми обґрунтовано

доцільність застосування нтелектуального підходу, що поєднує глибокий статичний аналіз вихідного коду з генеративними можливостями великих мовних моделей На основі проведеного аналізу визначено оптимальний технологічний стек для розробки програмного забезпечення:

1. Синтаксичний аналіз: використання абстрактних синтаксичних дерев (AST) для точного вилучення структури кодової бази (класів, функцій, методів).

2. Оркестрація та ШІ: застосування фреймворку LangChain для побудови гнучких конвеєрів обробки даних та методів інженерії підказок (Prompt Engineering) для формування точних контекстних запитів до мовних моделей.

3. Архітектура додатка: проєктування десктопного інтерфейсу з підтримкою асинхронності та багатопоточності для забезпечення стабільної роботи під час мережевих запитів. Застосування такого комплексного підходу дозволить автоматизувати рутинні процеси створення стандартизованої технічної документації (у форматі Markdown),що сприятиме суттєвому підвищенню ефективності роботи інженерних команд та мінімізації технічного боргу проєкту.

2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

У цьому розділі розглядається етап практичної розробки програмного забезпечення, призначеного для автоматизованої генерації технічної документації. Перехід від теоретичних досліджень до інженерної реалізації передбачає формування чітких функціональних та нефункціональних вимог, обґрунтування архітектури системи та проєктування внутрішніх структур даних.

2.1 Вимоги до системи

Розробка будь-якого програмного забезпечення на етапі проєктування вимагає ретельного системного аналізу та формування чіткого переліку вимог. Цей етап є критично важливим, оскільки саме він визначає функціональні межі майбутнього проєкту, його архітектурну складність та критерії якості розроблюваного продукту. Для забезпечення комплексного підходу до створення системи автоматизованої генерації технічної документації, усі висунуті до неї вимоги було класифіковано на дві основні категорії: функціональні, що описують безпосередню поведінку програми, та нефункціональні, які визначають технічні обмеження, архітектурні властивості та стандарти безпеки.

Функціональні вимоги формують ядро взаємодії розробника або технічного письменника із розробленою системою. У першу чергу, програмне забезпечення повинно забезпечувати зручний та гнучкий механізм завантаження вихідних даних. Система має підтримувати імпорт не лише окремих файлів вихідного коду, але й комплексне рекурсивне сканування цілих директорій проєкту. Наступним критичним аспектом є реалізація модуля багатомовного статичного аналізу. Для програмного коду, написаного мовою Python, система повинна використовувати глибокий синтаксичний парсинг за допомогою абстрактних синтаксичних дерев (AST), що гарантує максимальну точність вилучення інформації про класи та методи. Водночас для підтримки інших популярних мов програмування, таких як JavaScript, Java чи C#, передбачається застосування

лексичного аналізу на основі оптимізованих регулярних виразів. Окремою вагомою вимогою є забезпечення варіативності вихідних артефактів. Програма повинна надавати користувачу можливість вибору типу генерованого документа відповідно до поточних потреб розробки: від технічної специфікації прикладного інтерфейсу (API Reference) та головного файлу репозиторію (README) до загального архітектурного огляду проєкту та детального посібника користувача. Важливою функціональною особливістю є підтримка гібридної генерації, що передбачає здатність програми працювати у двох незалежних режимах. Основний інтелектуальний режим має використовувати потужності великих мовних моделей через фреймворк LangChain та зовнішні API для створення глибокого семантичного опису. Натомість резервний демонстраційний режим (Demo-режим) повинен гарантувати базову працездатність програми в умовах відсутності інтернет-з'єднання або лімітів доступу, генеруючи структуру документації на основі заздалегідь підготовлених шаблонів. Завершальною функціональною вимогою є механізм експорту результатів: згенеровані тексти мають зберігатися у форматі Markdown із додаванням необхідних метаданих або конвертуватися у повністю стилізовані HTML-сторінки з автоматичним підсвічуванням синтаксису коду.

Нефункціональні вимоги, своєю чергою, регламентують атрибути якості та інженерні обмеження системи. Пріоритетною вимогою є забезпечення асинхронності роботи додатка. Зважаючи на те, що звернення до зовнішніх мовних моделей через мережу інтернет можуть займати значний час, графічний інтерфейс програми не повинен блокуватися чи «зависати». Для забезпечення безперебійної взаємодії з користувачем усі обчислювальні процеси та мережеві запити мають бути обов'язково винесені в ізольовані фонові потоки.

Іншою надзвичайно важливою характеристикою є кросплатформенність розробленого рішення. Застосунок має гарантувати стабільну та ідентичну роботу в різних сімействах операційних систем, зокрема Windows, macOS та Linux. Виконання цієї вимоги покладається на використання кросплатформеної

мови програмування Python у поєднанні з портативним графічним фреймворком PyQt6. Нарешті, критичною складовою є безпека обробки конфіденційної інформації. Ключі доступу до API (API-ключі), які вводить користувач для авторизації, не повинні зберігатися у відкритому вигляді на накопичувачі, записуватися у системні логи програми або передаватися будь-яким стороннім телеметричним сервісам, окрім безпосередніх серверів постачальника обраної великої мовної моделі.

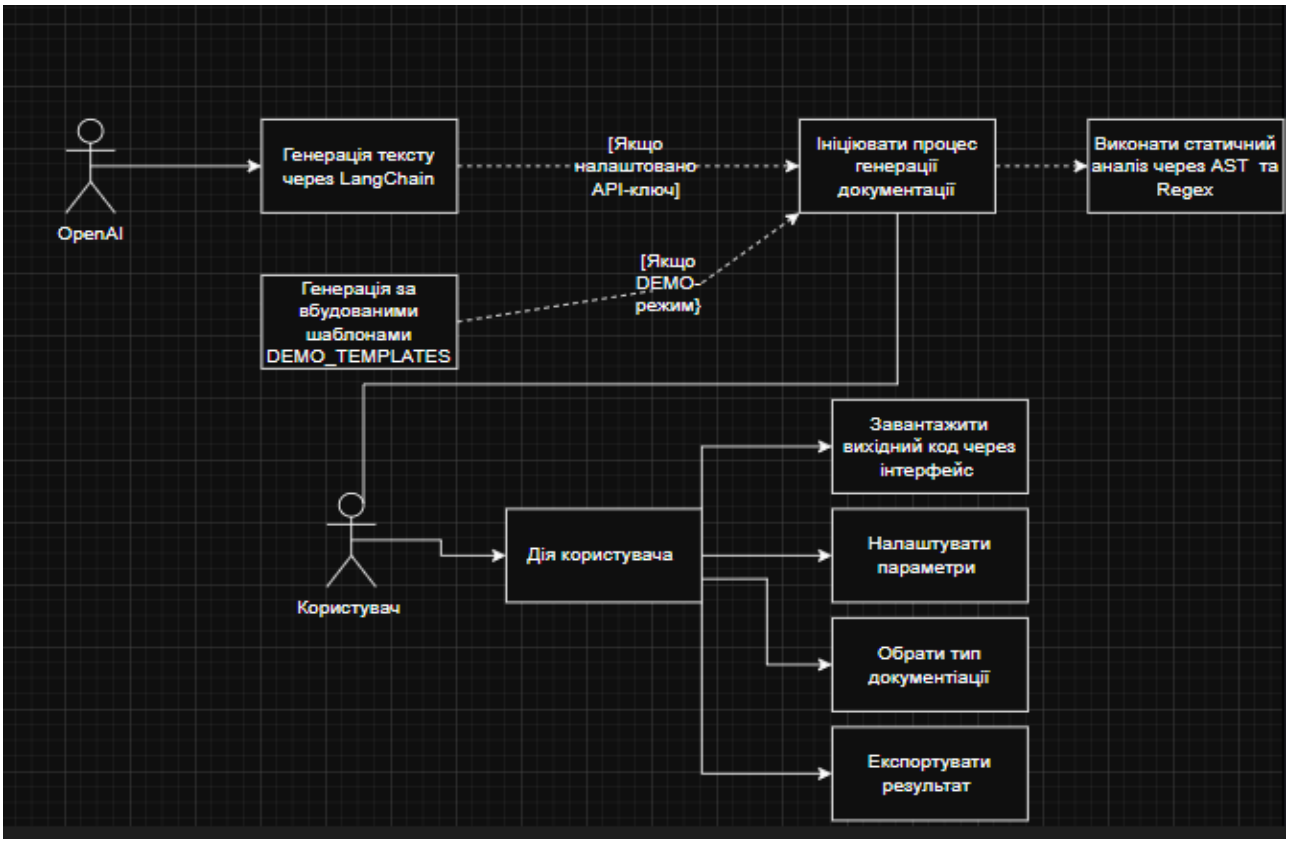

Рисунок 2.1 – Загальна архітектура системи

2.2 Архітектура системи

Архітектура розробленого програмного забезпечення базується на фундаментальних принципах інженерії програмного забезпечення, зокрема модульності, розподілу обов'язків та слабкої зв'язності компонентів. Для забезпечення високого рівня гнучкості, стабільності роботи та можливості

подальшого масштабування системи (наприклад, для майбутнього додавання підтримки нових мов програмування чи інтеграції нових нейромережевих моделей) мною було обрано багаторівневу архітектурну модель. Загалом уся логіка системи чітко розділена на п'ять взаємопов'язаних рівнів, кожен з яких інкапсулює специфічну бізнес-логіку. Верхнім рівнем архітектури виступає рівень представлення, який відповідає за безпосередню взаємодію з користувачем.

Цей шар реалізований у головному модулі програми за допомогою сучасного кросплатформеного фреймворку PyQt6. На цьому рівні відбувається обробка графічного інтерфейсу, перевірка та валідація вхідних конфігураційних налаштувань, а також управління візуальною зоною перетягування для зручного завантаження файлів вихідного коду. Згенеровані результати відображаються у спеціалізованому текстовому віджеті додатка. Важливим архітектурним рішенням на цьому етапі стала інкапсуляція всіх звернень до нижчих рівнів у фоновий асинхронний потік, що гарантує відсутність блокування чи «зависання» графічної оболонки під час тривалих системних обчислень.

Наступним, і чи не найважливішим, є рівень оркестрації, який виконує роль центрального ядра системи та безпосередньо керує всіма потоками даних. Цей компонент реалізований у вигляді окремого класу-генератора та спирається на структурний патерн проєктування «Фасад». Такий підхід надає єдиний уніфікований та максимально спрощений інтерфейс для рівня представлення, надійно приховуючи складність внутрішніх підсистем. Головне завдання оркестратора полягає у прийнятті сирих файлів вихідного коду, передачі їх до підсистеми аналізу, отриманні структурованих метаданих та подальшому їх спрямуванні у конвеєр штучного інтелекту. Крім того, на цьому рівні реалізовано механізм керування внутрішнім кешем, що дозволяє уникнути дублюючих мережевих запитів та суттєво оптимізувати використання ресурсів.

За первинну обробку вхідних даних відповідає рівень статичного аналізу, реалізований у відповідному спеціалізованому модулі парсингу. Його функціональне призначення полягає у локальному зчитуванні вихідного коду без

його безпосереднього виконання у середовищі. Для обробки файлів, написаних мовою Python, система використовує вбудовані засоби для глибокого розбору коду шляхом побудови абстрактних синтаксичних дерев (AST). Це гарантує найвищуточність вилучення інформації про структуру програми. Для обробки файлів інших підтримуваних мов програмування застосовуються оптимізовані лексичні аналізатори, побудовані на базі регулярних виразів.

Наступним логічним блоком є рівень інтеграції з великими мовними моделями, який відповідає за створення інтелектуального контенту. Цей шар повністю ізолює складну логіку взаємодії зі штучним інтелектом від решти програмних компонентів додатка. Він містить структуровану колекцію системних шаблонів та підказок для генерування різних типів технічної документації. Використання інструментарію фреймворку LangChain на цьому рівні дозволяє формувати гнучкі ланцюжки обробки даних, забезпечуючи стабільну комунікацію між результатами статичного аналізу та зовнішнім прикладним інтерфейсом обраної мовної моделі. Завершальним етапом обробки даних керує рівень форматування та експорту. До його складу входять спеціалізовані класи-експортери, які відповідають за фінальну стилізацію згенерованого тексту. На цьому етапі відбувається автоматичне додавання конфігураційних метаданих до готових документів формату Markdown, а також здійснюється їх програмна конвертація у повноцінні веб-сторінки формату HTML. Процес експорту супроводжується автоматичною інтеграцією механізмів підсвічування синтаксису, що забезпечує професійний, читабельний та естетичний вигляд фінальної технічної документації.

2.3 Проектування програми для налаштування параметрів

Для забезпечення гнучкості роботи розробленого програмного забезпечення та адаптації генерації під конкретні потреби розробника, було спроєктовано окрему підсистему конфігурації. Оскільки якість та специфіка згенерованої технічної документації безпосередньо залежить від параметрів

великої мовної моделі, користувачу необхідно надати зручний інструмент для керування цими налаштуваннями. Цей розділ логічно поділяється на структурування даних, розробку графічного інтерфейсу та забезпечення безпеки.

2.3.1 Структура конфігураційних даних

Для зберігання та передачі параметрів між графічним інтерфейсом та рівнем оркестрації (класом DocGenerator) використовується спеціальний клас даних. Це дозволяє уникнути передачі великої кількості окремих аргументів у функції та централізує керування налаштуваннями за замовчуванням. Основними параметрами, що підлягають конфігурації, є:

1. Ключ доступу (API Key): необхідний для авторизації запитів до сервісів OpenAI.

2. Модель (Model): вибір конкретної версії нейромережі (наприклад, gpt-4o mini або gpt-4-turbo), що дозволяє балансувати між швидкістю, вартістю та якістю аналізу.

3. Температура (Temperature): параметр у діапазоні від 0.0 до 1.0, який визначає рівень креативності моделі. Для технічної документації оптимальним є значення 0.3, що забезпечує детермінованість та точність тексту.

4. Максимальна кількість токенів (Max Tokens): ліміт обсягу згенерованого тексту для контролю витрат.

Лістинг 2.1 - Структура для зберігання конфігураційних параметрів програми

```
from dataclasses import dataclass

@dataclass
class AppConfig:
"""Клас для зберігання налаштувань генерації документації."""
api_key: str = ""
```

```
model_name: str = "gpt-4o-mini"

temperature: float = 0.3

max_tokens: int = 4096

def is_demo_mode(self) -> bool:

"""Перевіряє, чи працює програма в автономному режимі (без
 ключа)."""

return not bool(self.api_key.strip())
```

2.3.2 Графічний інтерфейс вікна налаштувань

Для взаємодії з користувачем було розроблено окреме модальне вікно на базі класу QDialog з бібліотеки PyQt6. Компонування елементів виконано за допомогою менеджера QFormLayout, який забезпечує акуратне вирівнювання віджетів у форматі "Мітка - Поле введення". Зовнішній вигляд розробленого вікна конфігурації наведено на рисунку 2.2.

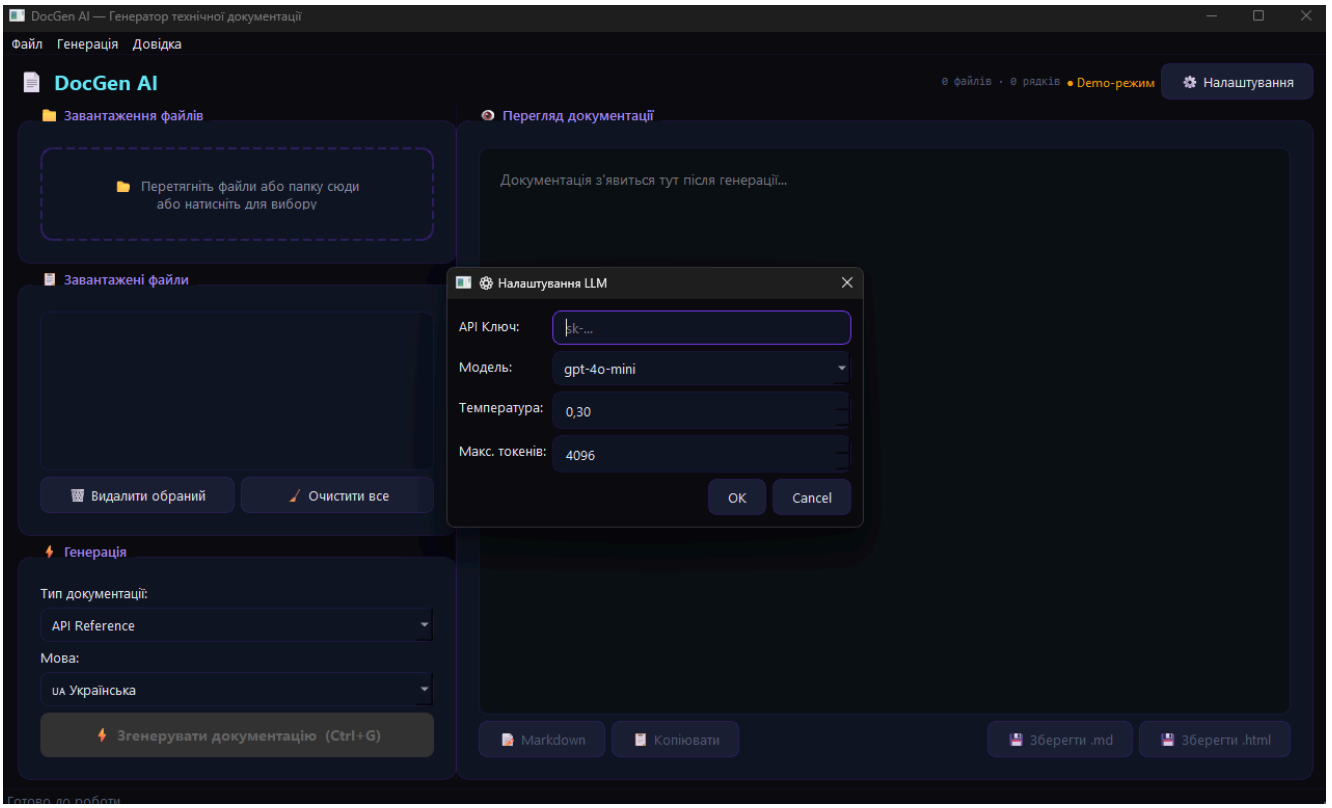

Рисунок 2.2 – Графічний інтерфейс вікна налаштувань параметрів генерації

Ключовою нефункціональною вимогою до системи є безпека облікових даних. Для виконання цієї вимоги у модулі налаштувань реалізовано наступні механізми: Ізоляція в пам'яті: API-ключ зберігається виключно в оперативній пам'яті як атрибут об'єкта AppConfig під час виконання програми. Він передається безпосередньо в ініціалізатор конвеєра LangChain і не записується у конфігураційні файли на жорсткому диску або в логи програми.

Автоматичний перехід у безпечний режим: якщо користувач залишає поле ключа порожнім, система автоматично ідентифікує це через метод is_demo_mode() та перемикає додаток на використання локальних шаблонів без здійснення мережевих запитів.

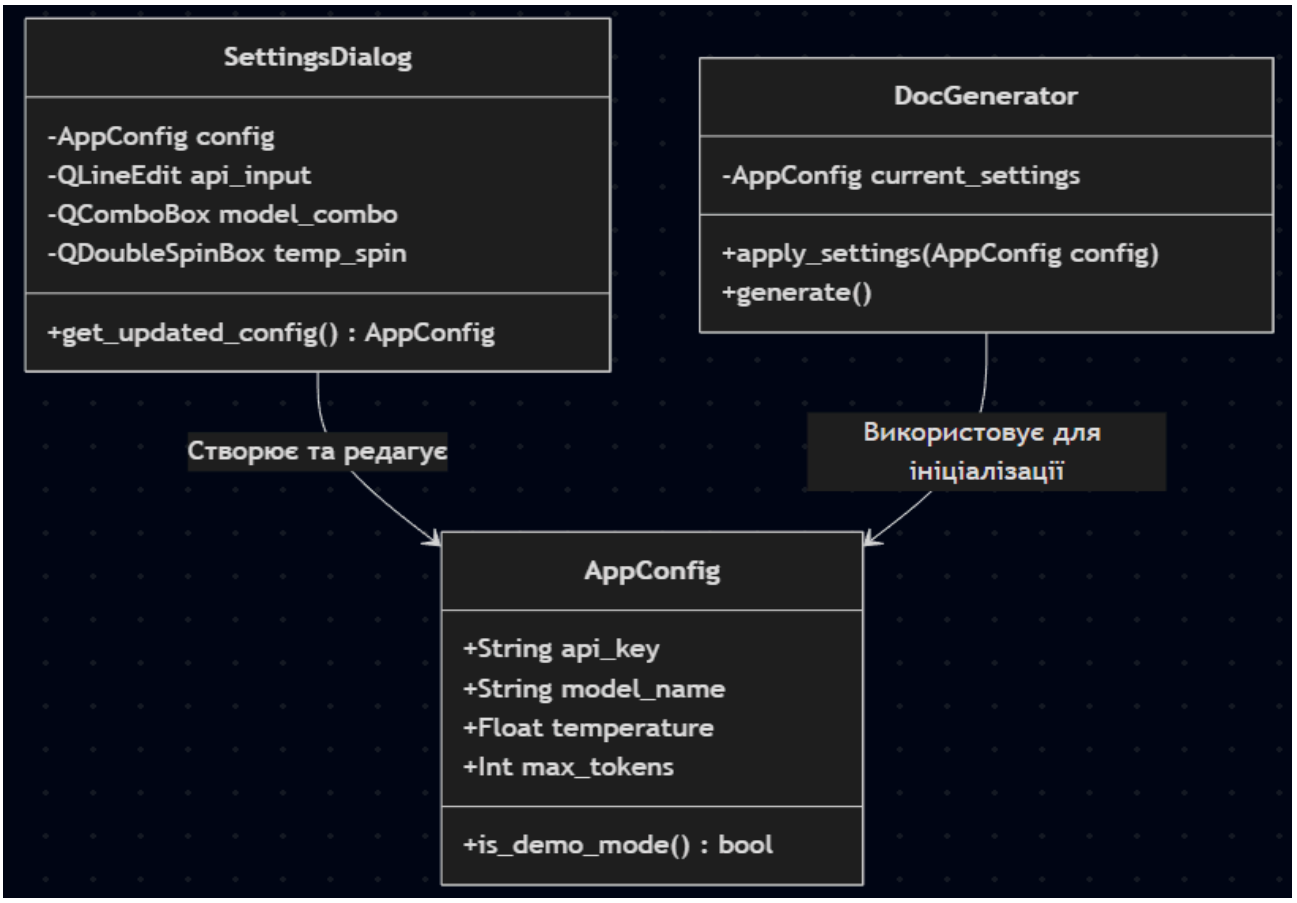

Рисунок 2.3 – Діаграма класів модуля налаштувань

2.4 Проєктування компоненту обміну даних

У процесі розробки десктопного програмного забезпечення я стикнувся з класичною проблемою проєктування графічних інтерфейсів: необхідністю розділення обчислювальної логіки та потоку відмальовування вікон Оскільки мережеві запити до прикладного інтерфейсу мовних моделей, а також процес статичного аналізу великих масивів коду можуть тривати від кількох секунд до хвилини, виконання цих операцій у головному потоці призвело б до повного "зависання" програми та операційна система могла б примусово її закрити.

Для вирішення цієї інженерної проблеми мною було спроєктовано спеціальний компонент обміну даних на базі багатопотокової архітектури. В якості основного інструменту я використав клас QThread з бібліотеки PyQt6 та механізм сигналів і слотів (Signals and Slots), який є стандартом для безпечної передачі даних між потоками. Основна ідея компоненту полягає у тому, що при натисканні кнопки генерації створюється ізольований фоновий потік. У цей потік передаються вхідні дані (шляхи до файлів та обраний тип документації). Після завершення роботи конвеєра LangChain, фоновий потік не намагається самостійно оновити графічний інтерфейс (що викликало б критичну помилку доступу до пам'яті), а генерує подію (сигнал), яка містить готовий текст формату Markdown. Головний потік перехоплює цей сигнал і безпечно виводить дані на екран. Програмну реалізацію компоненту асинхронного обміну даними наведено у лістингу 2.2.

Лістинг 2.2 – Реалізація фонового потоку генерації з використанням системи сигналів

```
from PyQt6.QtCore import QThread, pyqtSignal




class GenerateThread(QThread):
```

```
    """
    Фоновий потік для асинхронної генерації документації.
        Забезпечує обмін даними без блокування головного вікна програми.
      """     # Оголошення сигналів для безпечної передачі даних у головний потік     finished = pyqtSignal(str)   # Сигнал успішного завершення з текстом Markdown
       error  =  pyqtSignal(str)         # Сигнал помилки з текстом виключення


    def __init__(self, generator, files: list[str], doc_type: str):
        super().__init__()
        self.generator = generator  # Об'єкт класу DocGenerator
        self.files = files          # Список файлів для аналізу
         self.doc_type = doc_type    # Тип документації (README, API тощо)


    def run(self):
        """
        Головний метод, що виконується в ізольованому потоці.
        """
        try:
            # Виклик логіки генерації (аналіз AST -> LangChain)
```

```
                    result = self.generator.generate(self.files,
self.doc_type) # Безпечна відправка згенерованого результату в
інтерфейс

                self.finished.emit(result)          except Exception as
e:

                # У разі мережевої або синтаксичної помилки передаємо
опис

                self.error.emit(f"Помилка генерації: {str(e)}")
```

Для кращого розуміння життєвого циклу обміну даними між користувачем, графічним інтерфейсом, фоновим потоком та сервісом штучного інтелекту, я спроєктував UML-діаграму послідовності.

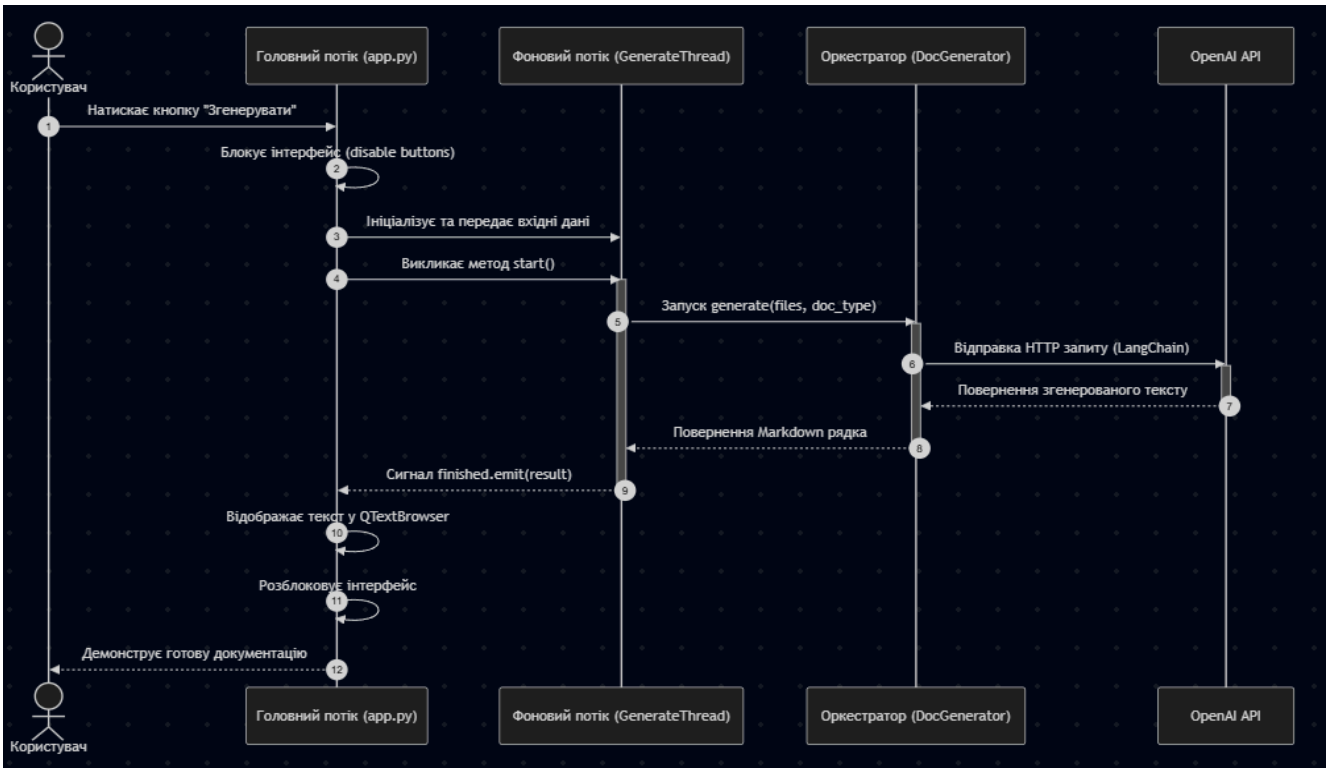

Рисунок 2.4 – Діаграма послідовності обміну даними під час генерації документації

2.5 Реалізація базових підсистем програми

Після завершення етапу проєктування архітектури та механізмів асинхронного обміну даними, наступним кроком стала безпосередня програмна реалізація ключових компонентів системи. Взаємодія користувача з розробленим програмним забезпеченням розпочинається у головному вікні. Саме тут реалізовано зону завантаження вихідного коду, вибір типу бажаної документації та управління процесом генерації. Інтерфейс спроєктовано таким чином, щоб мінімізувати кількість кроків для отримання фінального результату. Зовнішній вигляд головного вікна програми з доданими файлами вихідного коду наведено на рисунку 2.5.

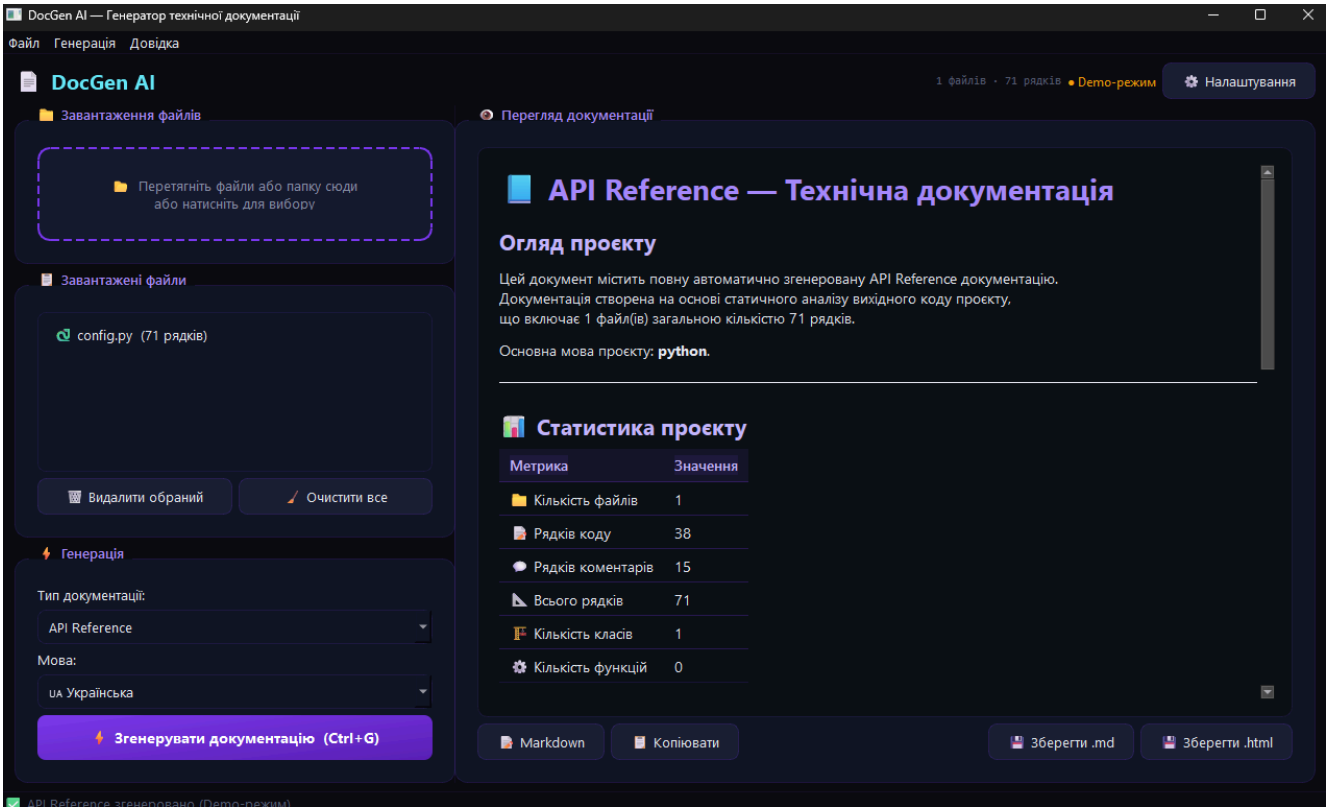

Рисунок 2.5 – Головне вікно розробленого програмного забезпечення

Цей розділ детально описує процес обробки завантажених даних: від написання коду для модуля статичного аналізу до налаштування інтеграційного конвеєра штучного інтелекту та реалізації підсистеми відображення і збереження результатів.

2.5.1 Реалізація модуля статичного аналізу вихідного коду

Фундаментом для якісної генерації документації є точне розуміння структури вихідного коду. Відправляти сирі файли безпосередньо у мовну модель неефективно через обмеження контекстного вікна та можливу втрату логічних зв'язків. Тому мною було реалізовано клас CodeAnalyzer у модулі analyzer.py, який виконує попередню обробку файлів. Для мови програмування Python реалізовано алгоритм на основі вбудованої бібліотеки ast. Цей підхід дозволяє перетворити текст програми на абстрактне синтаксичне дерево, обійти його вузли та безпечно вилучити назви класів, сигнатури методів, списки аргументів та існуючі коментарі розробника, не виконуючи сам код. Фрагмент реалізації методу аналізу файлів наведено у лістингу 2.4.

Лістинг 2.4 – Реалізація парсингу вихідного коду за допомогою абстрактних синтаксичних дерев

```
import ast

from dataclasses import dataclass


def _analyze_python_file(self, filepath: str) -> FileAnalysis:
"""Виконує глибокий статичний аналіз Python файлу через AST."""
with open(filepath, 'r', encoding='utf-8') as f:
source_code = f.read()


# Побудова синтаксичного дерева

tree = ast.parse(source_code)

analysis = FileAnalysis(filename=os.path.basename(filepath),
language="python")
```

```
# Обхід вузлів дерева

for node in ast.walk(tree):

if isinstance(node, ast.ClassDef):

# Вилучення інформації про клас
class_info = ClassInfo(

name=node.name,

docstring=ast.get_docstring(node)

)

analysis.classes.append(class_info)

elif isinstance(node, ast.FunctionDef):

# Вилучення інформації про функцію

func_info = FunctionInfo(

name=node.name,

docstring=ast.get_docstring(node),

args=[arg.arg for arg in node.args.args] )
analysis.functions.append(func_info)


return analysis
```

Для файлів інших форматів реалізовано спрощений лексичний аналіз за допомогою регулярних виразів, що робить розроблену систему універсальною.

2.5.2 Інтеграція з мовними моделями через конвеєр LangChain

Головним інтелектуальним ядром розробленого програмного забезпечення є модуль langchain_pipeline.py. Його завдання — прийняти структуровані дані від аналізатора, сформувати правильний контекст та безпечно відправити його до прикладного інтерфейсу великої мовної моделі. Для реалізації цього компоненту використано фреймворк LangChain. Замість класичних запитів, застосовано декларативний синтаксис мови виразів LangChain. Це дозволило об'єднати шаблон підказки, саму нейромережеву модель та обробник вихідних даних в єдиний послідовний ланцюжок за допомогою оператора конвеєризації. Код реалізації конвеєра наведено у лістингу 2.5.

Лістинг 2.5 – Створення конвеєра обробки даних

```
from langchain_openai import ChatOpenAI

from langchain_core.prompts import ChatPromptTemplate

from langchain_core.output_parsers import StrOutputParser

class LangChainPipeline:

    """

    Конвеєр для зв'язку між статичним аналізом та LLM.

    """    def __init__(self, api_key: str, model: str, temperature: float):

        # Ініціалізація клієнта OpenAI

        self.llm = ChatOpenAI(
```

```
            api_key=api_key,
            model=model,
            temperature=temperature
        )

    def generate_document(
        self,
        analysis_data: str,
        source_code: str,
        prompt_template: str
    ) -> str:
        """
            Створює ланцюжок викликів та генерує фінальний текст.
# Формування шаблону підказки з системними інструкціями
        prompt = ChatPromptTemplate.from_messages([
            ("system", prompt_template),
            (
```

```
            "human",
                    "Результати аналізу AST:\n{analysis}\n\nВихідний
код:\n{code}"
            )
        ])

        # Створення декларативного конвеєра
        chain = prompt | self.llm | StrOutputParser()
        # Виконання запиту та повернення чистого тексту Markdown
        return chain.invoke({
            "analysis": analysis_data,
            "code": source_code
        })
```

2.5.3 Реалізація підсистеми експорту та відображення результатів

Фінальним етапом роботи програми є відображення згенерованої документації безпосередньо у графічному інтерфейсі та її збереження у зручному для розробника форматі. Після успішного виконання фонового потоку генерації, текстовий результат у форматі Markdown передається до головного вікна програми та відображається у спеціалізованому текстовому віджеті. Це дозволяє розробнику одразу ознайомитися зі згенерованим описом класів та

методів, перевірити коректність архітектурних висновків нейромережі та, за необхідності, змінити налаштування генерації. Демонстрацію роботи системи із відображенням готового результату наведено на рисунку 2.7.

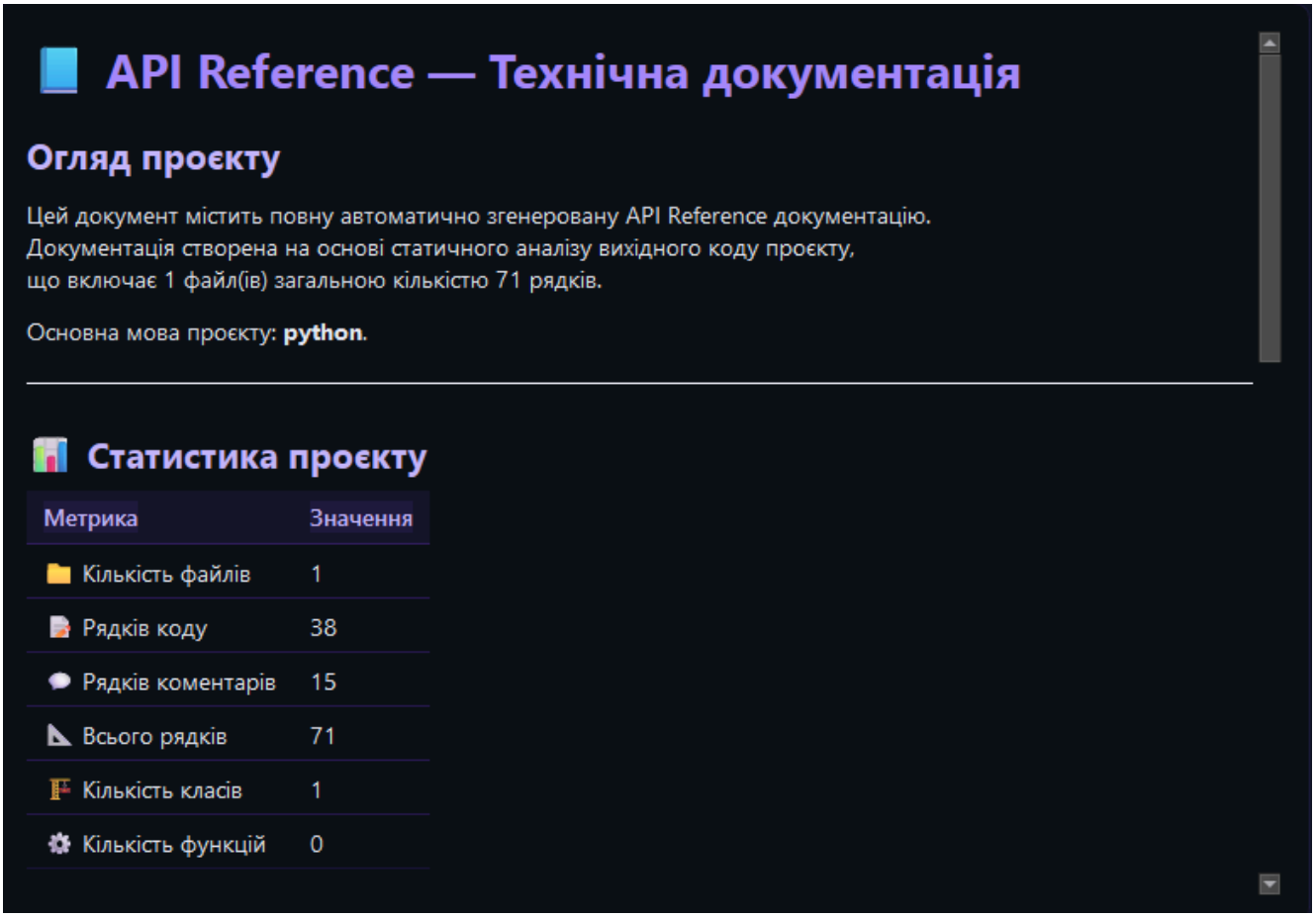

Рисунок 2.7 – Відображення згенерованої технічної документації у вікні програми

Для подальшого використання створених артефактів мною було реалізовано модуль exporters.py, який містить логіку конвертації та запису файлів на жорсткий диск. Згенерований текст за замовчуванням зберігається у форматі .md, при цьому програма автоматично додає на початок документа конфігураційний блок метаданих формату YAML, який фіксує дату створення та тип документації.

Для забезпечення можливості автономного перегляду документації поза

спеціалізованими редакторами, додатково реалізовано функцію конвертації Markdown у повноцінну HTML-сторінку. Під час експорту у згенерований HTML код ін'єктується кастомний файл стилів із темною темою оформлення та підключаються клієнтські скрипти для автоматичного підсвічування синтаксису у блоках прикладів коду. Це гарантує високу естетичну якість та професійний вигляд фінальної технічної документації.

2.5.4 Конструювання механізмів взаємодії з файловою системою

Важливим етапом конструювання програмного продукту стала реалізація зручного механізму імпорту вихідних даних. Оскільки технічна документація часто генерується не для одного файлу, а для цілих проєктів, стандартного діалогового вікна вибору файлів було недостатньо. Тому в основу взаємодії користувача з файловою системою було покладено технологію «Drag and Drop» . Для реалізації цього механізму у фреймворку PyQt6 було перевизначено базові події віджета QFrame, який виступає зоною завантаження. Під час перетягування об'єктів у вікно програми, система автоматично перехоплює подію dragEnterEvent, перевіряє MIME-тип даних (чи є вони локальними файлами) і надає дозвіл на їх скидання. Процес взаємодії користувача із зоною завантаження продемонстровано на рисунку 2.8.

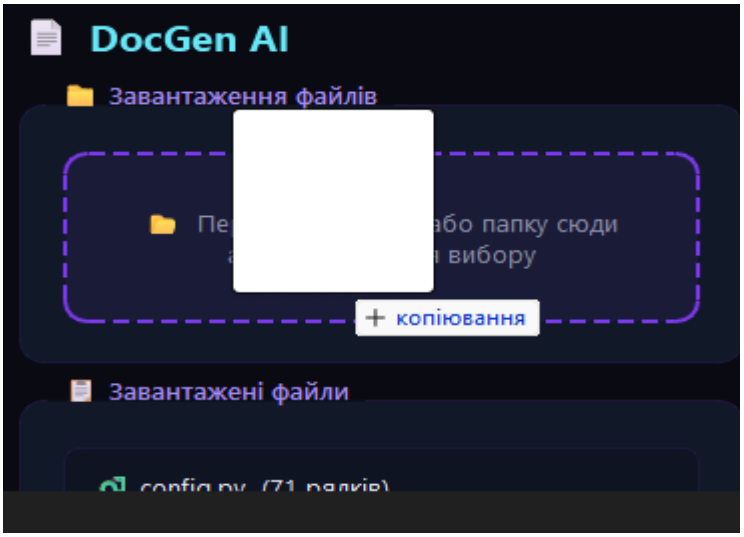

Рисунок 2.8 – Конструювання інтерактивної зони завантаження файлів методом Drag and Drop

Після відпускання кнопки миші спрацьовує подія dropEvent, яка запускає алгоритм рекурсивного сканування директорій. Оскільки проєкти містять багато службових файлів (зображення, бінарні збірки, конфігурації), алгоритм здійснює жорстку фільтрацію за розширеннями, залишаючи лише підтримувані формати вихідного коду (наприклад, .py, .js, .cpp). Фрагмент коду, що демонструє конструювання логіки обробки перетягування та фільтрації, наведено у лістингу 2.6.

Лістинг 2.6 – Реалізація обробки подій Drag and Drop та фільтрації файлів

```
def dragEnterEvent(self, event):
"""Перехоплення події перетягування файлів у
вікно.""" if event.mimeData().hasUrls():
 event.accept()
# Візуальний відгук (зміна кольору зони)
self.drop_frame.setStyleSheet("background-color: #2b2d30; border:
2px dashed #4a90e2;")
 else:
 event.ignore()def dropEvent(self, event):
 """Обробка відпускання файлів та їх рекурсивне сканування."""
self.drop_frame.setStyleSheet("") # Повернення базового стилю
valid_extensions = {'.py', '.js', '.java', '.cpp', '.cs',
'.ts'}
 for url in event.mimeData().urls():
 path = url.toLocalFile()
 if os.path.isdir(path):
 # Рекурсивний обхід директорії
 for root, _, files in os.walk(path):
```

```
for file in files:

if os.path.splitext(file)[1] in valid_extensions:

self.add_file_to_list(os.path.join(root, fil

 else:

# Обробка одиничного файлу

if os.path.splitext(path)[1] in valid_extensions:
self.add_file_to_list(path)
```

2.5.5 Конструювання системи сповіщень та обробки виключних ситуацій

Невід'ємною складовою конструювання надійного графічного інтерфейсу є розробка системи зворотного зв'язку з користувачем. Під час генерації документації можуть виникати різноманітні непередбачувані ситуації: відсутність вибраних файлів, обрив з'єднання з серверами OpenAI, або вичерпання ліміту токенів. Для коректного інформування розробника було сконструйовано централізовану систему обробки помилок на базі класу QMessageBox. Замість того, щоб програма аварійно завершувала роботу (Crash) при виникненні виключення Exception у конвеєрі LangChain, фоновий потік генерує спеціальнийсигнал, який перехоплюється головним вікном. Програмну реалізацію виклику діалогових вікон із попередженнями наведено у лістингу 2.7.

Лістинг 2.7 – Логіка конструювання діалогових вікон для сповіщення користувача

```
from PyQt6.QtWidgets import QMessageBox



def _show_error_message(
```

```
    self,
    title: str,
    text: str,   detailed_text: str = None
):
    """
    Генерація стандартизованого вікна з помилкою.
    """
    msg = QMessageBox(self)
    msg.setIcon(QMessageBox.Icon.Warning)
    msg.setWindowTitle(title)
    msg.setText(text)
    # Додавання технічних деталей для розробника
    if detailed_text:
        msg.setDetailedText(detailed_text)

    # Стилізація кнопок під загальну темну тему
    msg.setStyleSheet(
        """
        QLabel {
```

```
            color: white;
        }
        QPushButton {        background-color: #3c3f41;
        }
        """
    )

    msg.exec()
def on_generation_failed(self, error_traceback: str):
    """
    Слот, що реагує на помилку з фонового потоку LangChain.
    """
    # Розблокування кнопок
    self.set_ui_enabled(True)

    self._show_error_message(
        title="Помилка генерації",
        text=(
            "Виникла проблема під час звернення до мовної моделі. "
```

```
            "Перевірте API-ключ або інтернет-з'єднання."
        ),
        detailed_text=error_traceback)
```

2.6 Висновки до другого розділу

У другому розділі кваліфікаційної роботи було виконано повний цикл проєктування та програмної реалізації десктопної системи автоматизованої генерації технічної документації. Перехід від теоретичного огляду до практичної розробки розпочався з формування чітких функціональних та нефункціональних вимог, які стали фундаментальною базою для всіх подальших інженерних рішень. Для забезпечення високої надійності, масштабованості та зручності підтримки кодової бази було обрано та обґрунтовано багаторівневу модульну архітектуру. Такий підхід дозволив чітко розмежувати зони відповідальності між графічним інтерфейсом, ядром оркестрації, підсистемою статичного аналізу та конвеєром взаємодії зі штучним інтелектом. Важливим архітектурним рішенням стала реалізація гібридного режиму роботи, що дозволяє програмі функціонувати як з використанням зовнішніх великих мовних моделей, так і в автономному демонстраційному режимі.

У ході програмної реалізації було успішно вирішено низку складних завдань програмної інженерії:

Для забезпечення точного розпізнавання структури вихідного коду розроблено аналізатор, що базується на обході абстрактних синтаксичних дерев (AST) для мови Python та оптимізованому лексичному аналізі для інших мов програмування.

Інтеграцію з мовними моделями побудовано на базі оркестраційного фреймворку LangChain із застосуванням декларативного синтаксису LCEL, що забезпечило гнучке та надійне керування контекстом запитів.

Розроблено ергономічний кросплатформений графічний інтерфейс за допомогою бібліотеки PyQt6. Для уникнення блокування інтерфейсу під час тривалих обчислень та мережевих запитів спроєктовано надійну систему асинхронного обміну даними на основі ізольованих фонових потоків (QThread) та механізмів сигналів і слотів. Фінальним етапом стала реалізація підсистеми експорту, яка забезпечує зручне представлення результатів. Розроблене програмне забезпечення автоматично формує документацію (описи API, файли README, архітектурні огляди) та експортує її у формат Markdown із конфігураційними метаданими, а також конвертує у стилізовані HTML-сторінки з підсвічуванням синтаксису. Підсумовуючи, спроєктована та реалізована система повністю відповідає сформованим на початку розділу вимогам. Обраний технологічний стек та архітектурні патерни дозволили створити швидкодіючий, безпечний та зручний інструмент розробника, який готовий до етапу тестування та практичного впровадження.

3 ТЕСТУВАННЯ

Забезпечення якості програмного продукту є завершальним та критично важливим етапом життєвого циклу розробки. Оскільки розроблена система автоматизованої генерації технічної документації призначена для використання інженерами та технічними письменниками, її графічний інтерфейс та внутрішні алгоритми обробки даних повинні бути максимально надійними та відмовостійкими. Для перевірки відповідності розробленого додатка вимогам, визначеним на етапі архітектурного проєктування, мною було обрано методологію функціонального тестування методом «чорного ящика». Цей підхід дозволяє перевірити коректність роботи програми через взаємодію з її графічним інтерфейсом, імітуючи реальну поведінку кінцевого користувача, без безпосереднього втручання у вихідний код модулів.

3.1 План тестування системи

Для забезпечення комплексного покриття всіх можливих сценаріїв використання розробленого програмного забезпечення, мною було складено детальний план функціонального тестування. Головною метою цього плану є систематична та послідовна перевірка реакції системи як на правильні дії користувача, так і на некоректні вхідні дані або непередбачувані системні збої.

Процес тестування був логічно розділений на кілька взаємопов'язаних етапів. На першому етапі передбачалася перевірка графічного інтерфейсу та механізмів завантаження вхідних даних. Особлива увага приділялася працездатності зони перетягування файлів та коректності роботи алгоритмів фільтрації за розширеннями. Другий етап був присвячений тестуванню підсистеми конфігурації, де перевірялися механізми валідації числових полів та безпека обробки конфіденційних даних, зокрема ключів доступу до

прикладного інтерфейсу.

На третьому етапі планувалося масштабне тестування безпосередньоїбізнес-логіки генерації документації. Цей крок включав перевірку коректності роботи багатомовного парсера, формування запитів до великих мовних моделей та верифікацію згенерованого тексту відповідно до обраного шаблону. Завершальним, четвертим етапом стало тестування обробки виключних ситуацій, що передбачало штучне моделювання критичних умов, таких як обрив мережевого з'єднання або спроба обробки пошкоджених файлів, з метою перевірки стійкості додатка до аварійних завершень.

3.2 Тестування застосунку

Виконання плану тестування розпочалося з перевірки базового користувацького сценарію, а саме процедури імпорту файлів вихідного коду та налаштування параметрів мовної моделі. Під час тестування модуля завантаження було успішно верифіковано роботу механізму інтерактивного перетягування. При спробі перемістити у вікно програми цілу директорію проєкту, застосунок коректно проініціював рекурсивне сканування, автоматично відкинув медіафайли, виконувані файли та скомпільовані бібліотеки, залишивши у списку обробки виключно текстові файли з вихідним кодом. Елементи керування цим списком, такі як видалення окремих елементів або повне очищення черги, відпрацювали без жодних затримок чи помилок доступу до пам'яті.

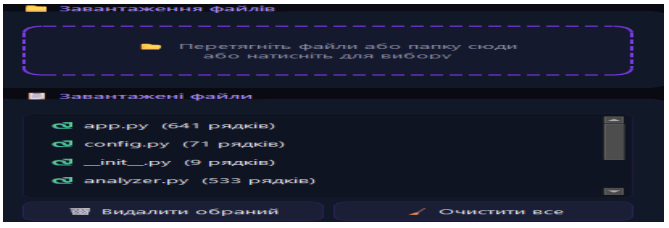

Рисунок 3.1 – Відображення успішно завантажених файлів вихідного коду у

графічному інтерфейсі

Наступним кроком стала перевірка модального вікна конфігурації параметрів генерації. Мною було проведено серію тестів на стійкість до некоректного введення. Спроби ввести літери або спеціальні символи у поля налаштування температури моделі та ліміту токенів були успішно заблоковані на рівні графічних віджетів. Поле введення ключа доступу продемонструвало коректну поведінку щодо маскування введених символів, що унеможливлює випадкову компрометацію облікових даних (Рисунок 3.2).

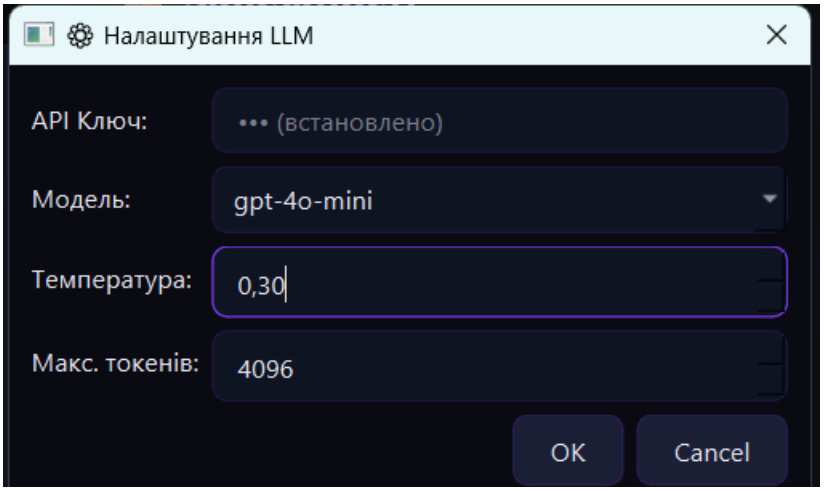

Рисунок 3.2 – Тестування валідації вхідних даних у вікні конфігурації параметрів моделі

Крім того, тестування підтвердило надійність резервного механізму: у разі запуску генерації з порожнім полем ключа, програма не видає помилку авторизації, а плавно перемикається у демонстраційний режим, використовуючи вбудовані шаблони для створення базової структури документації.

Окремо було протестовано результати роботи конвеєра генерації. Після завантаження еталонних файлів вихідного коду та ініціалізації процесу створення документації у форматі специфікації прикладного інтерфейсу, система успішно передала дані через ланцюжок LangChain. Аналіз отриманого результату показав високу точність вилучення інформації: згенерований текст містив правильні назви класів, описи методів та їхніх аргументів. Підсистема експорту також пройшла перевірку без зауважень, коректно зберігаючи результати у форматах Markdown із

додаванням метаданих та генеруючи валідні HTML-сторінки з інтегрованим стилістичним оформленням та підсвічуванням синтаксису.

3.3 Тестування компонента обміну даними

Найбільш критичним з точки зору стабільності роботи десктопних застосунків є тестування багатопотокової архітектури, яка відповідає за обмін даними між графічним інтерфейсом та фоновими обчислювальними процесами. Оскільки статичний аналіз об'ємних проєктів та очікування відповіді від віддалених серверів штучного інтелекту вимагають значного часу, ці процеси були ізольовані у фонових потоках. Для перевірки надійності цього архітектурного рішення мною було проведено серію функціональних стрес-тестів.

У першу чергу перевірялася загальна чуйність інтерфейсу під час тривалих обчислювальних навантажень. Було ініційовано процес генерації масивного обсягу документації, який тривав понад п'ятнадцять секунд. Протягом усього цього часу я активно взаємодіяв з головним вікном програми: переміщував його по робочому столу, згортав та розгортав. Графічна оболонка працювала абсолютно плавно, операційна система не фіксувала стан зависання, що повністю підтверджує правильність реалізації асинхронного виконання завдань.

Наступним кроком стала перевірка механізмів запобігання стану гонитви та захисту від дублюючих викликів. Одразу після натискання користувачем кнопки старту генерації, система миттєво переводила цю кнопку та інші ключові елементи керування у неактивний стан. Цей запобіжний захід успішно заблокував можливість повторного або багаторазового випадкового кліку, який міг би призвести до запуску десятків паралельних потоків, переповнення оперативної пам'яті та критичного перевитрачання фінансових лімітів на використання хмарного API.

Завершальним випробуванням для компонента обміну даними стало тестування його поведінки у нестандартних ситуаціях, пов'язаних із мережевими збоями. Під час активної фази звернення системи до сервісів мовної моделі було навмисно розірвано з'єднання з глобальною мережею інтернет. Архітектурапрограми успішно впоралася з цією виключною ситуацією: замість аварійного падіння застосунку, фоновий потік коректно перехопив помилку тайм-ауту з'єднання та безпечно передав сигнал у головний потік.

У результаті програма автоматично розблокувала всі елементи керування та вивела на екран інформативне діалогове вікно, яке пояснило користувачу причину зупинки процесу.

3.4 Висновки до третього розділу

У третьому розділі було проведено ретельне функціональне тестування розробленого програмного забезпечення для автоматизованої генерації технічної документації. Базуючись на заздалегідь складеному комплексному плані, мною було перевірено всі ключові сценарії взаємодії системи з кінцевим користувачем: від початкового імпорту та фільтрації вихідних файлів до фінального експорту готових текстових артефактів.

Проведені випробування графічного інтерфейсу повністю підтвердили його ергономічність, інтуїтивну зрозумілість та високий рівень захисту від некоректних дій користувача. Всі візуальні компоненти та елементи керування функціонують стабільно та передбачувано. Перевірка внутрішньої бізнес-логіки продемонструвала високу точність алгоритмів статичного аналізу коду, правильність формування контекстних запитів для великих мовних моделей та безпомилкову роботу резервного демонстраційного режиму. Окремої уваги заслуговують результати стрес-тестування багатопотокового компонента обміну даними. Штучно змодельовані затримки обчислень та імітації обривів мережевого з'єднання наочно довели, що застосована архітектура надійно захищає програму

від блокувань і зависань. Механізми обробки виключень своєчасно та коректно інформують користувача про поточний стан виконання завдань, не допускаючи аварійних завершень роботи.

Підсумовуючи результати всіх етапів тестування, можна з упевненістю констатувати, що розроблений програмний продукт є стабільним, повністю відповідає функціональним та нефункціональним вимогам, сформованим на етапі проєктування, не містить критичних архітектурних чи логічних дефектів і є цілком готовим до практичного впровадження у реальні процеси інженерії програмного забезпечення.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

Безпека життєдіяльності та охорона праці утворюють фундаментальну основу будь-якої сучасної виробничої або науково-технічної діяльності, виступаючи гарантом збереження найвищої соціальної цінності — життя та здоров'я людини. У широкому розумінні безпека життєдіяльності вивчає загальні закономірності виникнення небезпек, їхні властивості, наслідки впливу на організм людини, а також основи захисту від них у будь-яких умовах перебування. Охорона праці, своєю чергою, є більш вузькою та спеціалізованою галуззю, що являє собою комплексну систему правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини саме у процесі її трудової діяльності. Актуальність цих питань у сучасному світі невпинно зростає через стрімкий науково-технічний прогрес, впровадження новітніх технологій, використання складного обладнання та нових синтетичних матеріалів, які часто несуть у собі приховані загрози для людського організму.

З правової точки зору, система охорони праці базується на конституційних гарантіях права кожного громадянина на належні, безпечні та здорові умови праці. Законодавча база чітко регламентує обов'язки роботодавця щодо створення безпечного робочого середовища, впровадження сучасних засобів безпеки, запобігання професійним захворюванням та виробничому травматизму. Крім того, державна політика у сфері охорони праці ґрунтується на принципах пріоритетності життя і здоров'я працівників по відношенню до результатів виробничої діяльності підприємства. Це означає, що жодні економічні вигоди, виконання планів чи досягнення високих виробничих показників не можуть виправдати ризик втрати здоров'я або життя персоналу. Відповідальність за дотримання цих норм покладається безпосередньо на керівництво, яке зобов'язане

забезпечити фінансування заходів з охорони праці, регулярне навчання персоналу, проведення інструктажів та медичних оглядів.

Соціально-економічне значення охорони праці важко переоцінити, оскільки безпечні умови праці безпосередньо впливають на продуктивність і загальну ефективність підприємства. Створення комфортного робочого середовища знижуєвтомлюваність, підвищує концентрацію уваги, зменшує кількість помилок і брак у роботі. З іншого боку, ігнорування вимог безпеки призводить до значних економічних втрат, які складаються з виплат за листками непрацездатності, компенсацій за шкоду, заподіяну здоров'ю, витрат на ремонт пошкодженого внаслідок аварій обладнання, а також втрат від простоїв виробництва. Таким чином, інвестиції у заходи з охорони праці є не лише виконанням юридичних зобов'язань, але й економічно доцільним кроком, який забезпечує стабільність, конкурентоспроможність та позитивний імідж підприємства на ринку праці. Для досягнення цієї мети на підприємствах впроваджуються системи управління охороною праці, що передбачають безперервний моніторинг умов праці, ідентифікацію небезпек, оцінку ризиків та розробку ефективних превентивних заходів.

4.1 Динамічні явища на поверхні землі

Літосфера Землі перебуває у стані постійної динаміки, що зумовлено як внутрішніми (ендогенними), так і зовнішніми (екзогенними) геологічними процесами. Динамічні явища на поверхні землі — це природні процеси, які призводять до раптових або поступових змін рельєфу, деформації земної кори та часто супроводжуються катастрофічними наслідками для населення, інфраструктури та екосистем. До найбільш поширених та небезпечних динамічних явищ належать землетруси, зсуви, селі, снігові лавини та карстові провалля.

Землетруси є одними з найбільш руйнівних ендогенних динамічних явищ. Вони являють собою раптові підземні поштовхи та коливання земної поверхні, що виникають внаслідок раптового вивільнення накопиченої пружної енергії у надрах Землі (найчастіше на межах тектонічних плит). Ступінь руйнівної дії землетрусів оцінюється за 12-бальною макросейсмічною шкалою (MSK-64 або її аналогами). На території України сейсмічно небезпечними є зони Карпат (зокрема, вплив зони Вранча), Кримських гір та південно-західного узбережжя Чорного моря. Під час проєктування будь-яких інженерних споруд, включно з центрами обробки даних (ЦОД) для IT-інфраструктури, обов'язково враховується рівень сейсмічної небезпеки регіону.

Зсуви — це ковзне зміщення мас гірських порід вниз по схилу під дією сили тяжіння. Зсувні процеси найчастіше провокуються підмиванням схилів річками чи морем, перезволоженням ґрунту внаслідок інтенсивних опадів, таненням снігу, а також нераціональною господарською діяльністю людини (наприклад, підрізанням схилів під час будівництва, знищенням рослинності, надмірним поливом територій). Зсуви становлять серйозну загрозу для інженерних комунікацій, ліній електропередач та оптоволоконних мереж, які прокладаються у гірській або горбистій місцевості.

Селі (селеві потоки) — це раптові, короткочасні, але надзвичайно потужні потоки суміші води, ґрунту, піску, каміння та уламків гірських порід, що виникають у басейнах гірських річок. Вони характеризуються високою швидкістю руху (до 10-15 м/с) та величезною руйнівною силою. Основною причиною їх виникнення є сильні зливи, інтенсивне танення льодовиків або прориви гірських озер. В Україні селенебезпечними є гірські райони Карпат та Криму.

Снігові лавини являють собою стрімке сповзання або падіння мас снігу з гірських схилів. Зсув снігової маси відбувається, коли сила тяжіння перевищує сили зчеплення снігу зі схилом. Лавини можуть руйнувати будівлі, перекривати транспортні магістралі та лінії зв'язку.

Карстові явища пов'язані з процесом розчинення гірських порід (вапняку, гіпсу, крейди, кам'яної солі) поверхневими або підземними водами. Внаслідок

цього під землею утворюються порожнини (печери, канали), що з часом може призвести до раптового провалу земної поверхні. В Україні карстові процеси особливо розвинені на території Поділля (зокрема в Тернопільській області), Криму та Донбасу. Це вимагає проведення ретельних геологорозвідувальних робіт перед початком будь-якого капітального будівництва.

Зниження ризиків від динамічних явищ на поверхні землі вимагає впровадження комплексу превентивних заходів. До них належать постійний сейсмологічний та геологічний моніторинг, інженерний захист територій (будівництво підпірних стінок, селевловлювачів, дренажних систем), суворе дотримання будівельних норм у небезпечних зонах, а також розробка планів евакуації та ліквідації наслідків надзвичайних ситуацій. Для забезпечення безперебійної роботи програмного забезпечення та збереження критично важливих даних в умовах можливих природних катастроф застосовується географічний розподіл серверних потужностей та регулярне резервне копіювання інформації (Disaster Recovery).

4.2 Особливості заходів електробезпеки на підприємствах

Електрична енергія є базовою складовою функціонування будь-якого сучасного підприємства, особливо у сфері інформаційних технологій, де безперебійна робота обчислювальної техніки є критично важливою. Проте використання електроустановок завжди пов'язане з ризиком ураження працівників електричним струмом. Дія електричного струму на організм людини має складний і різнобічний характер: термічний (опіки, нагрівання тканин), електролітичний (розклад крові та інших органічних рідин) та біологічний (подразнення і збудження живих тканин, що може призвести до зупинки серця або дихання).

Відповідно до нормативних актів з охорони праці (зокрема, ПУЕ — Правил улаштування електроустановок), за ступенем небезпеки ураження електричним струмом усі виробничі приміщення поділяються на три категорії:

1. Приміщення без підвищеної небезпеки.

2. Приміщення з підвищеною небезпекою.
3. Особливо небезпечні приміщення.

Стандартні робочі місця інженерів-програмістів та ІТ-фахівців (офісні кабінети) зазвичай класифікуються як приміщення без підвищеної небезпеки (сухі, з нормальною температурою та ізолюючими підлогами). Проте спеціалізовані приміщення, такі як серверні кімнати або апаратні вузли, можуть належати до приміщень з підвищеною небезпекою через наявність великої кількості металевого обладнання, потужних систем кондиціонування та струмопровідних антистатичних фальшпідлог.

Для запобігання електротравматизму на підприємствах впроваджується комплекс технічних та організаційних заходів. До основних технічних заходів захисту належать:

Робоча та подвійна ізоляція струмопровідних частин комп'ютерної техніки та периферійних пристроїв, що унеможливлює випадковий дотик до струмоведучих елементів.

Захисне заземлення — навмисне електричне з'єднання з землею або її еквівалентом металевих неструмопровідних частин обладнання (наприклад, корпусів системних блоків, стійок серверів), які можуть опинитися під напругою внаслідок пробою ізоляції.

Занулення — електричне з'єднання металевих частин електроустановок із глухозаземленою нейтраллю джерела живлення, що забезпечує швидке спрацювання автоматичних вимикачів у разі короткого замикання.

Пристрої захисного відключення (ПЗВ) — швидкодіючі автоматичні вимикачі, які миттєво (за долі секунди) знеструмлюють ділянку мережі у разі виявлення струму витоку, що виникає, коли людина торкається оголеного дроту.

Особливості електробезпеки безпосередньо для розробників програмного забезпечення полягають у правильній експлуатації офісної техніки. Забороняється

використання пошкоджених кабелів живлення, перевантаження розеток великою кількістю приладів через подовжувачі-двійники, а також самостійне відкриття корпусів системних блоків або моніторів без попереднього відключення їх від електромережі. Важливим елементом архітектури безпеки ІТ-підприємств є використання систем безперебійного живлення (ДБЖ), які не лише захищають обладнання від перепадів напруги, але й мінімізують ризик виникнення пожежонебезпечних коротких замикань.

Організаційні заходи електробезпеки на підприємстві включають сувору регламентацію допуску до роботи. Усі працівники, незалежно від посади, зобов'язані проходити вступний та періодичні (повторні) інструктажі з охорони праці. Крім того, персонал має бути навчений правилам безпечної експлуатації закріпленого за ними комп'ютерного обладнання та алгоритмам надання першої долікарської допомоги у разі ураження колег електричним струмом (вміти швидко та безпечно знеструмити постраждалого та провести реанімаційні заходи до приїзду швидкої допомоги).

4.3 Висновки до четвертого розділу

На основі проведеного аналізу можна зробити висновок, що забезпечення безпеки життєдіяльності та охорони праці є невіддільною складовою успішного виробничого процесу. Пріоритет життя та здоров'я працівника над будь-якими економічними результатами вимагає постійного моніторингу умов праці та регулярної атестації робочих місць. Це дозволяє своєчасно ідентифікувати весь комплекс шкідливих і небезпечних факторів — фізичних, хімічних, біологічних та психофізіологічних, здатних призвести до виробничого травматизму чи професійних захворювань. Зокрема, глибокий руйнівний вплив вібрації на серцево-судинну та нервову системи організму яскраво доводить необхідність застосування багаторівневого захисту від фізичних загроз. Тільки синергія сучасних інженерно-технічних рішень (зменшення небезпеки у самому джерелі),

суворого дотримання регламентованих режимів праці та відпочинку, використання надійних засобів індивідуального захисту і регулярного медичного контролю здатна мінімізувати виробничі ризики та гарантувати персоналу безпечні й комфортні умови життєдіяльності.

ВИСНОВКИ

У кваліфікаційній роботі було успішно вирішено актуальне завдання сучасної програмної інженерії — автоматизацію процесу створення технічної документації до вихідного коду. Виконаний комплекс теоретичних досліджень та практичної розробки дозволив створити повноцінний програмний продукт, який суттєво
зменшує витрати часу розробників на рутинний опис архітектури, класів та методів.

У процесі роботи було спроєктовано гнучку багаторівневу архітектуру, що забезпечила слабку зв'язність компонентів та високу масштабованість системи. Фундаментом програми став розроблений модуль статичного аналізу, який за допомогою обходу абстрактних синтаксичних дерев (AST) та лексичного аналізу безпечно вилучає структурну інформацію з вихідних файлів. Інтелектуальним ядром системи виступив інтеграційний конвеєр на базі фреймворку LangChain. Його використання дозволило організувати надійну взаємодію з великими мовними моделями, забезпечивши точне формування контексту, безпечне управління обліковими даними та можливість роботи в резервному автономному режимі.

Для забезпечення комфортної роботи кінцевого користувача було розроблено ергономічний кросплатформений графічний інтерфейс із застосуванням бібліотеки PyQt6. Завдяки впровадженню асинхронного обміну даними та багатопотоковості, програма зберігає високу чуйність навіть під час тривалих обчислень та мережевих запитів. Фінальним етапом обробки стала реалізація підсистеми експорту, яка автоматично генерує документацію у форматі Markdown із відповідними метаданими та конвертує її у стилізовані HTML-сторінки з підсвічуванням синтаксису коду.

Проведене функціональне тестування підтвердило високу надійність реалізованих алгоритмів, коректну обробку мережевих збоїв та загальну відмовостійкість системи. Мета роботи досягнута повною мірою: розроблений

додаток є стабільним та ефективним інструментом, що повністю відповідає поставленим вимогам і готовий до впровадження у реальний процес створення програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1.Методичні вказівки до виконання кваліфікаційної роботи магістра для здобувачів спеціальності 121 – Інженерія програмного забезпечення, всіх форм навчання / укладачі: Михалик Д.М., Цуприк Г.Б., Бревус В.М., Мудрик І.Я. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. – 44 с. URL: http://elartu.tntu.edu.ua/handle/lib/50316

2.Модуль ast — Abstract Syntax Trees [Електронний ресурс]. – URL: https://docs.python.org/3/library/ast.html

3.Модуль ast — Abstract Syntax Trees [Електронний ресурс]. – URL: https://docs.python.org/3/library/ast.html

4.Офіційна документація бібліотеки PyQt6 [Електронний ресурс]. – URL: https://www.riverbankcomputing.com/static/Docs/PyQt6/

5.Офіційна документація мови програмування Python [Електронний ресурс]. – URL: https://docs.python.org/3/

6.Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Addison-Wesley Professional, 2021. 448 p.

7.Brown T. et al. Language Models are Few-Shot Learners // Advances in Neural Information Processing Systems. – 2020. – Vol. 33. – P. 1877-1901.

8.Bubeck S. et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4 // arXiv preprint arXiv:2303.12712. – 2023. – URL: https://arxiv.org/abs/2303.12712

9.Doxygen: Source code documentation generator tool [Electronic resource]. – URL: https://www.doxygen.nl/index.html

10.Fowler M. Refactoring: Improving the Design of Existing Code. 2nd ed. Addison-Wesley Professional, 2018. 448 p.

11.Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0 / ed. H. Washizaki. IEEE Computer Society, 2024. 411 p.

12.Hou X. et al. Large Language Models for Software Engineering: A Systematic

14.Literature Review // ACM Transactions on Software Engineering and Methodology. – 2024. – URL: https://arxiv.org/abs/2308.10620

15.LangChain Documentation [Electronic resource]. – URL: https://python.langchain.com/docs/get_started/introduction

16.Markdown Guide: Basic Syntax [Electronic resource]. – URL: https://www.markdownguide.org/basic-syntax/

17.Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall, 2017. 432 p.

18.OpenAI API Reference [Electronic resource]. – URL: https://platform.openai.com/docs/api-reference

19.Percival H., Gregory B. Architecture Patterns with Python: Enabling Test-Driven 20.Development, Domain-Driven Design, and Event-Driven Microservices. O'Reilly Media, 2020. 306 p.

21.QThread Class | Qt Core 6.7.0 [Electronic resource]. – URL: https://doc.qt.io/qt-6/qthread.html

22.Sphinx Documentation Generator [Electronic resource]. – URL: https://www.sphinx-doc.org/en/master/

23.Андрейчук Н.І. Охорона праці : навч. посіб. / Н.І. Андрейчук, Ю.В. Кіт, С.В. Шибанов, О.В. Шерстньова. Львів : Видавництво Львівська політехніка, 2021. 276 с.

24.Купчик М.П., Гандзюк М.П., Степанець І.Ф. та ін. Основи охорони праці. – К.: Основа, 2000. 416 с.

25.Желібо Є.П. Безпека життєдіяльності : підручник / В. В. Зацарний. Київ : Каравела, 2023. 344 с.

26.Атаманчук П.С. Безпека життєдіяльності: навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

ДОДАТКИ

ДОДАТОК А

Тези конференції

IX Міжнародна студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

УДК 004.8:004.42
Бойко В.
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ТЕХНІЧНОЇ ДОКУМЕНТАЦІЇ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ LANGCHAIN

Науковий керівник: Стоянов Ю. М., канд. техн. наук, доцент каф. ПІ

Boiko V. R.
Ternopil Ivan Puluj National Technical University

DEVELOPMENT OF SOFTWARE FOR AUTOMATED GENERATION OF TECHNICAL DOCUMENTATION USING THE LANGCHAIN FRAMEWORK

Supervisor: Stoianov Yu. M., PhD, Associate Professor of the SE Department

Ключові слова: LangChain, Python, автоматизація документації, великі мовні моделі, LLM, RAG, технічна документація, розробка ПЗ.
Keywords: LangChain, Python, documentation automation, Large Language Models, LLM, RAG, technical documentation, software development.

Вступ. Сучасна розробка програмного забезпечення характеризується високою швидкістю змін та складністю архітектурних рішень. Одним із найбільш критичних, але водночас рутинних етапів є створення технічної документації. Традиційні засоби генерації фокусуються лише на структурі коду, не пояснюючи високорівневу логіку. Метою роботи є створення системи, що за допомогою ШІ здатна «розуміти» контекст коду та формувати документацію у форматі Markdown.

Основна частина та архітектура. Архітектура рішення базується на використанні фреймворку LangChain, який виступає сполучною ланкою між кодом та LLM. Процес включає завантаження файлів проекту, їх семантичну сегментацію (Text Splitters) та створення векторних представлень для бази даних (Chroma/FAISS). Завдяки механізму RAG, система формує відповіді, спираючись на актуальний контекст сирцевого коду. Використання ланцюжків (Chains) дозволяє послідовно обробляти складні залежності між модулями, що забезпечує цілісність опису всієї архітектури.

Висновки. Розроблене ПЗ демонструє значне скорочення часу на підготовку документації (до 60–70%) порівняно з ручним написанням. Застосування LangChain дозволяє гнучко налаштовувати систему під різні мови програмування. Такий підхід забезпечує відповідність документації фактичному стану коду, що є критично важливим для динамічних IT-проектів.

ДОДАТОК Б

Посилання на репозиторій GitHub

https://github.com/dismember12/Generating-technical-documentation-using-the-LangChain-framework