МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
„КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. Сікорського”

Факультет прикладної математики

Кафедра програмного забезпечення комп’ютерних систем

КУРСОВА РОБОТА

з дисципліни «Компоненти програмної інженерії. Частина 4. Якість та тестування програмного забезпечення»

на тему

Розробка програмного продукту «Перевірка документації ІТ-проєкту на атомарність та двозначність»

Виконав студент

ІІІ курсу групи КП-21

Семенюк Сергій

Керівник роботи

доцент, Ткаченко К.О.

Оцінка

_____

(дата, підпис)

КИЇВ 2024

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ 3

ВСТУП 5

1. ТЕОРЕТИЧНА ЧАСТИНА 7

1.1. Визначення атомарності та двозначності в документації ІТ-проєктів та принципи її перевірки. 7

1.2. Роль атомарності та двозначності в якості документації ІТ-проєкту та її вплив на ефективність реалізації проєкту. 7

1.3. Процеси та інструменти для автоматизації перевірки документації ІТ-проєкту на атомарність та двозначність. 9

1.4. Об’єктно-орієнтоване програмування та його застосування у розробці програмного продукту. 11

2. ОПИС ПРОГРАМИ 13

2.1. Функціональні характеристики. 13

2.2. Опис використаних бібліотек. 13

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ 18

3.1. Обґрунтування вибору інструментів для розробки програмного забезпечення для перевірки атомарності та двозначності документації. 18

3.2. Пояснення фрагментів коду. 21

3.3. Опис результатів роботи програми. 35

ВИСНОВКИ 39

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 40

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ ТА ТЕРМІНІВ

Web - (англ. World Wide Web) Глобальний інформаційний простір, заснований на фізичній інфраструктурі Інтернету і протоколі передачі даних HTTP;

MVC – (англ. Model View Controller) архітектурний шаблон проєктування, який використовується для розробки програмного забезпечення з розділенням відповідальностей між компонентами системи.

IT – (англ. Information Technology) галузь, яка відповідає за збір, зберігання і передачу інформації за допомогою технічних пристроїв і, в цілому, за спілкування людей на відстані.

ASP.NET – фреймворк для розробки веб-додатків, що використовує .NET технології для створення серверних додатків.

JavaScript – мова програмування, що використовується для додавання інтерактивних елементів на веб-сторінках.

OpenXml – бібліотека для роботи з документами Microsoft Office, зокрема для маніпулювання файлами формату .docx в .NET.

HTTP – протокол передачі гіпертексту, який використовується для обміну даними між клієнтом і сервером у веб-додатках.

API – інтерфейс програмування додатків, набір функцій, що дозволяють взаємодіяти з іншими програмами чи системами.

CSS – каскадні таблиці стилів, мова розмітки для визначення вигляду веб-сторінок.

Bootstrap — CSS-фреймворк для створення адаптивних та мобільних веб-інтерфейсів, який включає готові стилі, компоненти та шаблони для швидкої розробки інтерфейсів користувача.

jQuery — бібліотека JavaScript, яка спрощує маніпуляції з DOM та реалізацію взаємодії з користувачем на веб-сторінках.

.docx — формат файлу документа Microsoft Word, що використовується для збереження текстових документів.

Atomicity — принцип у документації IT-проєктів, що означає, що вимога не повинна бути розділена на менші частини без втрати змісту та важливості.

Ambiguity — двозначність, що стосується неясності або невизначеності в трактуванні термінів чи вимог в документації.

ВСТУП

Дана курсова робота присвячена розробці програмного продукту для перевірки атомарності та двозначності в документації IT-проєктів. Завдяки використанню мови програмування C# і фреймворку ASP.NET, розроблене програмне забезпечення забезпечує високу швидкодію та ефективність, що є особливо важливим для обробки великих документів у форматі .docx. Такий підхід дозволяє досягти значно кращих результатів порівняно з іншими мовами програмування, наприклад PHP або Python, які часто використовуються для подібних завдань. Тому розроблене програмне забезпечення є дійсно актуальним. Дана тематика курсової роботи обрана через актуальність перевірки документації на атомарність та двозначність у процесах розробки IT-проєктів. Зростаюча складність проектів вимагає високої якості технічної документації, а відсутність чітко визначених вимог може призвести до значних проблем у розробці. Тому важливим є створення програмного забезпечення, яке автоматизує ці процеси, дозволяючи забезпечити точність і зрозумілість документації, що сприяє ефективності та успіху проєктів.

Об'єктом дослідження є процес перевірки завантаженого документа на атомарність та двозначність, визначення його формату та типу, а також генерування коментарів для покращення його якості на основі результатів перевірки.

Метою роботи є розроблення програмного забезпечення для перевірки документів на атомарність та двозначність, а також генерації відповідних коментарів для покращення якості документації на основі результатів перевірки.

До функціональних можливостей програми належать: читання та аналіз завантаженого користувачем документа, перевірка на атомарність, перевірка на двозначність, можливість вибору типу перевірок за допомогою галочок,

додавання коментарів до документа, можливості завантажити модифікований файл.

Розроблене програмне забезпечення може бути використане тестувальниками для забезпечення якості документації в IT-проєктах. Інструмент дозволяє автоматизувати перевірку документації, що сприяє уникненню помилок у розробці та забезпеченню чітких, зрозумілих вимог

Пояснювальна записка складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел (6-ти найменувань, з них 2 – іноземною мовою). Робота містить 7 рисунків. Загальний обсяг роботи – 30 друкованих сторінок, з них 29 сторінки основного тексту та 1 сторінка списку використаних джерел.

1. ТЕОРЕТИЧНА ЧАСТИНА

1.1. Визначення атомарності та двозначності в документації ІТ-проєктів та принципи її перевірки.

Атомарність у контексті документації ІТ-проєктів є принципом, за яким кожна вимога або інструкція повинна бути представлена окремо, без можливості подальшого поділу на менші частини без втрати її значення чи цілісності. Атомарні вимоги повинні бути чіткими, самодостатніми та не повинні містити зайвих залежностей або необхідності для додаткового пояснення. Це забезпечує зрозумілість і ефективність виконання задач, що значно знижує ймовірність непорозумінь під час розробки.

Двозначність, у свою чергу, означає наявність термінів або фраз, що можуть мати декілька інтерпретацій або можуть бути зрозумілі по-різному різними людьми. Двозначні вимоги можуть призвести до помилок під час розробки, адже кожен учасник проєкту може трактувати їх по-своєму. Це знижує ефективність роботи команди та підвищує ризик появи непотрібних або неправильно реалізованих функцій.

1.2. Роль атомарності та двозначності в якості документації ІТ-проєкту та її вплив на ефективність реалізації проєкту.

Атомарність та двозначність у документації ІТ-проєктів є основними факторами, які значною мірою впливають на успіх і ефективність реалізації проєкту. Їх значення виходить далеко за межі теоретичних понять, оскільки безпосередньо визначає, як швидко і якісно буде виконана розробка, тестування та впровадження програмного продукту.

Атомарність документації дозволяє кожній вимозі бути чітко визначеною і зрозумілою. Коли вимоги атомарні, вони містять тільки один аспект, що дозволяє команді зосередитися на конкретних завданнях без додаткових узагальнень чи перехресних залежностей. Це полегшує реалізацію програмного коду, тестування, а також допомагає у комунікації між усіма учасниками проєкту. Ретельно сформульовані атомарні вимоги скорочують час на роз'яснення і уточнення, що безпосередньо підвищує продуктивність і зменшує ймовірність помилок. Вони дозволяють розробникам, тестувальникам і менеджерам зосередитися на конкретних частинах системи без ризику нерозуміння або невірної інтерпретації.

Двозначність у документації, навпаки, є джерелом плутанини і суперечок. Наявність нечітких або багатозначних термінів в документах призводить до різних інтерпретацій, що може негативно вплинути на якість роботи команди. Різночитання вимог між різними учасниками проєкту (наприклад, між замовником і розробниками або між різними командами розробників) може спричинити непорозуміння, затримки у виконанні завдань та збільшення витрат на виправлення помилок. Коли терміни не визначені однозначно, це ускладнює процес прийняття рішень і може призвести до дублювання зусиль, або навіть до розробки некоректного функціоналу, який не відповідає вимогам кінцевих користувачів або замовника.

Таким чином, забезпечення атомарності та усунення двозначностей у документації є невід'ємними елементами успішного і ефективного виконання ІТ-проєкту. Правильно оформлені вимоги дозволяють швидше досягти кінцевих цілей проєкту, зменшити витрати часу і ресурсів на роз'яснення та виправлення помилок, а також забезпечити більшу узгодженість між усіма учасниками проєкту.

1.3. Процеси та інструменти для автоматизації перевірки документації ІТ-проєкту на атомарність та двозначність.

Автоматизація процесів перевірки документації на атомарність та двозначність є важливим етапом у забезпеченні якості ІТ-проєктів. Для ефективної автоматизації необхідно використовувати спеціалізовані інструменти та алгоритми, що дозволяють здійснити перевірку документації без втручання людини та з високою точністю.

Процес автоматизованої перевірки

1. Завантаження документа: Спочатку необхідно завантажити документ для перевірки. Це може бути .docx, .pdf або .txt файл. Інтерфейс надає можливість завантаження файлів через механізм перетягування або вибору файлу за допомогою кнопки.
2. Обрання типів перевірки: Користувач обирає, чи хоче перевіряти атомарність, двозначність, чи обидва параметри. Цей процес здійснюється через інтерактивні чекбокси в інтерфейсі, що дозволяють вибрати відповідні опції перевірки.
3. Аналіз документа:
    - Перевірка на атомарність передбачає аналіз того, чи є кожен елемент документа чітко сформульованим та не допускає розподілу на частини без втрати змісту.
    - Перевірка на двозначність полягає у виявленні неясних, двозначних фраз чи термінів, які можуть мати кілька інтерпретацій.
4. Автоматична обробка та додавання коментарів: Після вибору параметрів перевірки система автоматично обробляє документ та додає коментарі до тексту, вказуючи на можливі проблеми з атомарністю або двозначністю.

5. Завантаження обробленого файлу: Після завершення перевірки користувач може завантажити документ з доданими коментарями, які вказують на проблемні ділянки документа.

Інструменти для автоматизації

- OpenXml: Бібліотека, що дозволяє працювати з .docx файлами, що є основним форматом для перевірки атомарності. OpenXml забезпечує доступ до вмісту документа, дозволяючи здійснювати пошук та модифікацію текстових елементів.
- RegEx (Регулярні вирази): Для перевірки на двозначність можна використовувати регулярні вирази, які дозволяють ефективно виявляти шаблони тексту, що містять потенційно двозначні фрази.
- Алгоритми обробки тексту: Для аналізу на атомарність можна використовувати алгоритми, що перевіряють, чи є речення або вимоги документа занадто складними чи занадто широкими, щоб їх можна було розбити на окремі частини без втрати змісту.
- ASP.NET Core: Веб-фреймворк, що використовується для створення користувацького інтерфейсу, дозволяючи завантажувати файли, вибирати параметри перевірки та обробляти файли на сервері.
- JavaScript: Використовується для динамічного завантаження документів на веб-сторінку, обробки подій користувацького інтерфейсу, таких як перетягування файлів або вибір перевірок. JavaScript також взаємодіє з серверною частиною через API для завантаження результатів перевірки.

Автоматизація перевірки документації дозволяє значно зменшити час на виконання рутинних задач, покращити точність виявлення помилок у

документації, а також забезпечити безперервний процес перевірки в межах IT-проєкту.

1.4. Об’єктно-орієнтоване програмування та його застосування у розробці програмного продукту.

Об’єктно-орієнтоване програмування (ООП) – це методологія, яка змінила підхід до розробки програмного забезпечення, роблячи його більш структурованим та інтуїтивно зрозумілим. Основна ідея ООП полягає у використанні об’єктів – структур даних, що включають в себе як атрибути, так і поведінку (методи). Це дозволяє розробникам моделювати реальний світ у програмному коді, що спрощує розуміння та управління складними системами.

Однією з ключових концепцій ООП є інкапсуляція, яка дозволяє обмежувати доступ до даних об’єкту та керувати ним через визначені інтерфейси. Це підвищує безпеку даних та гнучкість коду, оскільки зміни в одній частині програми не впливають безпосередньо на інші. Наслідування дає змогу створювати нові класи на основі вже існуючих, розширюючи їхню функціональність та перевикористовуючи код, що сприяє економії часу та зусиль. Поліморфізм, ще одна важлива властивість ООП, дозволяє об’єктам різних класів використовувати один і той же інтерфейс, що забезпечує гнучкість та здатність коду до адаптації в різних ситуаціях.

Застосування принципів ООП у розробці програмних продуктів сприяє створенню модульного, легко масштабованого та підтримуваного коду. Це особливо важливо в умовах сучасної швидкоплинної IT-індустрії, де програмні продукти постійно еволюціонують та адаптуються до змінних потреб користувачів і ринку. Використання ООП дозволяє ефективно

управляти складністю проєктів, спрощуючи процеси розробки та тестування, що є ключовими факторами успішності будь-якого програмного продукту.

2. ОПИС ПРОГРАМИ

2.1. Функціональні характеристики.

Розроблене програмне забезпечення дозволяє користувачу перевіряти документацію IT-проєкту на атомарність та двозначність через веб-інтерфейс. Користувач може завантажити файл .docx документа, який після аналізу буде перевірений на ці параметри. Програмне забезпечення автоматично додає коментарі до документа, вказуючи на атомарність та двозначність його змісту. Результати перевірки надаються користувачу у вигляді модифікованого файлу, який можна завантажити.

2.2. Опис використаних бібліотек.

1) OpenXML

OpenXML - це бібліотека для роботи з документами Office Open XML, що дозволяє маніпулювати файлами .docx, .xlsx та .pptx без необхідності використовувати Microsoft Office. В цьому проєкті вона використовується для обробки та редагування .docx файлів, що є основним форматом для перевірки атомарності та двозначності в документації IT-проєктів.

Особливості OpenXML:

Легко створюються нові документи або редагуються існуючі .docx файли, додаються текст, таблиці, зображення, а також форматується текст, що дозволяє перевіряти структуру документа на атомарність і двозначність.

Забезпечує доступ до структурних елементів документа (параграфи, таблиці, стилі тощо), що дозволяє детально аналізувати та змінювати його вміст.

Підтримує додавання коментарів та анотацій до окремих частин тексту, що дає змогу реалізувати систему коментування для перевірки атомарності та двозначності.

Можна виконувати складні операції форматування, такі як налаштування стилів, шрифтів, кольорів та меж, що дозволяє створювати професійно оформлені звіти та рекомендації.

Підтримка обробки великих документів без необхідності використання Microsoft Word, що дозволяє здійснювати автоматизовану перевірку документації та генерувати звіти.

Переваги використання OpenXML у проєкті:

- Інтуїтивно зрозумілий API: Легкий доступ до структури .docx файлів для ефективної роботи з текстом, таблицями та коментарями.
- Часова ефективність: Автоматизує перевірку та прискорює обробку документації, генеруючи звіти з коментарями швидко.

OpenXML підходить для різноманітних завдань, пов’язаних з обробкою документації, включаючи перевірку атомарності та двозначності, автоматизацію аналізу та генерування коментарів у .docx файлах. Ця бібліотека особливо корисна для розробників .NET, які хочуть інтегрувати функціональність перевірки документації без необхідності використання Microsoft Word.

2) jQuery

jQuery - це популярна бібліотека JavaScript, яка спрощує маніпуляції з DOM (Document Object Model) і реалізацію взаємодії з користувачем на

веб-сторінках. У цьому проєкті jQuery використовується для динамічного оброблення завантаження файлів, попереднього перегляду документів, а також для реалізації інтерфейсу користувача (UI), включаючи функціональність перетягування файлів (drag-and-drop).

Особливості jQuery:

Спрощення роботи з DOM: jQuery дозволяє швидко маніпулювати елементами HTML, додавати ефекти та реагувати на події користувача за допомогою простого синтаксису.

Анімації та ефекти: jQuery підтримує вбудовані анімаційні ефекти, які покращують взаємодію користувача з веб-сторінкою, роблячи її більш інтерактивною.

Події: бібліотека спрощує обробку подій, таких як кліки, перетягування, зміна значень тощо, що дозволяє створювати зручні інтерфейси для користувачів.

Переваги використання jQuery у проєкті:

- Зручність: дозволяє швидко та ефективно реалізовувати складні динамічні інтерфейси з мінімумом коду, що значно спрощує розробку.
- Крос-браузерність: jQuery забезпечує стабільну роботу на всіх популярних браузерах, що є важливим для підтримки користувачів з різними налаштуваннями.
- Швидкість розробки: із великою кількістю готових рішень та плагінів, jQuery дозволяє швидко додавати функціональність до веб-інтерфейсів.

- Взаємодія з іншими бібліотеками: jQuery без проблем інтегрується з іншими бібліотеками та фреймворками, що дозволяє розширювати можливості веб-додатків.

jQuery є важливим інструментом для покращення інтерфейсу користувача в цьому проєкті, зокрема для реалізації функцій завантаження файлів, попереднього перегляду та динамічних змін елементів інтерфейсу, таких як кнопки, форми та панелі.

3) Bootstrap

Bootstrap - це популярний CSS-фреймворк для створення адаптивних та мобільних веб-інтерфейсів. Він включає набір готових стилів, компонентів та шаблонів для швидкої розробки привабливих інтерфейсів користувача.

Особливості Bootstrap:

Адаптивність: Забезпечує автоматичну підлаштування макету під різні розміри екрану, що є важливим для сучасних веб-додатків.

Широкий набір компонентів: Включає кнопки, форми, таблиці, панелі, модальні вікна, навігаційні елементи тощо.

CSS та JavaScript: Bootstrap поєднує стильові рішення з вбудованими JavaScript компонентами для інтерактивності.

Переваги використання Bootstrap у проєкті:

- Прискорює розробку завдяки готовим рішенням: Bootstrap включає безліч готових стилів, компонентів та шаблонів, що дозволяє значно скоротити час на розробку користувацьких інтерфейсів.
- Підтримка крос-браузерності: Bootstrap забезпечує стабільну роботу на всіх популярних браузерах, що забезпечує зручність використання інтерфейсу на різних пристроях.
- Простота в інтеграції та налаштуванні під ваші потреби: Легко інтегрується з іншими бібліотеками та фреймворками. Bootstrap дозволяє швидко налаштовувати стилі та компоненти, адаптуючи їх до специфічних вимог проєкту, зокрема через свою системи класів та змінних для стилізації.

Таким чином, Bootstrap є важливим інструментом для створення зручних та функціональних веб-інтерфейсів, що дозволяє швидко адаптувати інтерфейси під різні пристрої та екрани.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1. Обґрунтування вибору інструментів для розробки програмного забезпечення для перевірки атомарності та двозначності документації.

1) C#

Мова програмування C# є ідеальним вибором для розробки програмного продукту "Перевірка документації IT-проєкту на атомарність та двозначність". Вона надає потужні інструменти для обробки даних і маніпулювання файлами, що особливо важливо для роботи з документами у форматі .docx. Завдяки бібліотеці OpenXml, C# дозволяє ефективно працювати з документами Word, здійснюючи перевірку атомарності та недвозначності тексту.

C# є частиною .NET Framework, який забезпечує зручну інтеграцію з іншими інструментами та технологіями для веб-розробки через ASP.NET. Це дозволяє створювати як веб-додатки для перевірки документації, так і можливість обробки запитів та відповіді в реальному часі через API.

Мова має потужну систему типів, що знижує ймовірність помилок при обробці даних, і автоматичне управління пам’яттю, що дозволяє знизити кількість потенційних помилок, пов’язаних із управлінням ресурсами. Це особливо важливо при обробці великих документів .docx, оскільки забезпечує стабільність і ефективність роботи програми.

2) ASP .NET Core Web App

Вибір ASP.NET Core Web App для розробки програмного забезпечення для перевірки документації IT-проєкту на атомарність та двозначність є обґрунтованим і має низку переваг, що роблять цей фреймворк ідеальним рішенням для такого типу проєктів.

ASP.NET Core є крос-платформенним, що дозволяє розгортати веб-додаток на різних операційних системах, таких як Windows, Linux та macOS. Це забезпечує гнучкість у виборі хостингових середовищ та дає можливість працювати з різними конфігураціями серверів.

ASP.NET Core відомий своєю високою продуктивністю. Завдяки оптимізаціям і поліпшенням в ядрі фреймворку, веб-додатки на ASP.NET Core виконуються швидше порівняно з багатьма іншими веб-фреймворками.

Модульність ASP.NET Core дає можливість використовувати лише необхідні компоненти для функціональності перевірки атомарності та двозначності документації. Це знижує навантаження на систему, що важливо для масштабованих додатків, які можуть обробляти велику кількість файлів одночасно. Також це спрощує тестування та розгортання програмного забезпечення.

За допомогою ASP.NET Core можна створити зручний та інтуїтивно зрозумілий інтерфейс для користувачів за допомогою стандартних веб-технологій, таких як HTML, CSS, JavaScript. Це дозволяє користувачам завантажувати документи, вибирати параметри перевірки (атомарність, двозначність) та отримувати результати у вигляді коментарів до файлів, що додаються до документа.

ASP.NET Core також надає потужні засоби для забезпечення безпеки додатку, такі як аутентифікація, авторизація та захист від CSRF-атак, що важливо для гарантування безпеки файлів користувачів під час завантаження та обробки документів на сервері.

Таким чином, використання ASP.NET Core Web App дозволяє створити масштабоване, ефективне та безпечне програмне забезпечення для перевірки документації ІТ-проєкту на атомарність і двозначність.

3) MVC

Вибір MVC (Model-View-Controller) архітектури для розробки програмного забезпечення, особливо для таких задач, як перевірка документації ІТ-проєкту на атомарність та двозначність, є обґрунтованим. MVC є популярним архітектурним інструментом у розробці веб-додатків.

MVC розділяє додаток на три основні компоненти: модель (дані), вигляд (представлення даних) і контролер (логіка обробки). Це розділення дозволяє легше управляти комплексністю додатку, оскільки кожна частина відповідає за свій власний аспект програми.

Завдяки розділенню відповідальності, в MVC додатках легше вносити зміни або додавати нові функції. Наприклад, інтерфейс користувача може бути змінений без необхідності переписування бізнес-логіки.

MVC підтримує масштабування та розширення, що є важливим для додатків, які можуть рости або змінюватися з часом.

MVC спрощує процес модульного та інтеграційного тестування. Кожен компонент може бути тестований окремо, що забезпечує високу якість коду та легкість у виявленні помилок.

3.2. Пояснення фрагментів коду.

Тут описано основні методи контролера PagesController, які відповідають за обробку документів, їх перевірку на атомарність і двозначність, а також за додавання коментарів до документів.

Метод UploadDocument з контролера:

```
public async Task<IActionResult> UploadDocument(IFormFile file, bool atomicityCheck, bool ambiguityCheck)
{
    if (file == null || file.Length == 0)
    {
        return Json(new { success = false, message = "Файл не був вибраний." });
    }

    if (file.ContentType != "application/vnd.openxmlformats-officedocument.wordprocessingml.document")
    {
        return Json(new { success = false, message = "Неправильний тип файлу. Потрібен файл формату .docx" });
    }

    var tempFilePath = Path.Combine(Path.GetTempPath(), file.FileName);

    try
    {
        using (var stream = new FileStream(tempFilePath, FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }

        var checkedFileName = Path.GetFileNameWithoutExtension(file.FileName) + "_checked.docx";
        var checkedFilePath = Path.Combine(Path.GetTempPath(), checkedFileName);

        AddCommentToDocument(tempFilePath, checkedFilePath, atomicityCheck, ambiguityCheck);

        var fileBytes = await System.IO.File.ReadAllBytesAsync(checkedFilePath);

        System.IO.File.Delete(tempFilePath);
        System.IO.File.Delete(checkedFilePath);

                        return File(fileBytes, "application/vnd.openxmlformats-officedocument.wordprocessingml.document",
checkedFileName);
    }
    catch (Exception ex)
    {
        return StatusCode(500, $"Сталася помилка: {ex.Message}");
    }
}
```

Цей метод контролера обробляє завантаження файлів, їх перевірку на атомарність та двозначність, а також додавання коментарів до документа. Спочатку перевіряється наявність файлу та його тип (.docx). Якщо файл коректний, він копіюється в тимчасову папку, а потім генерується нове ім'я для обробленого файлу з суфіксом _checked. Викликається метод

AddCommentToDocument, який додає коментарі на основі вибраних перевірок атомарності та двозначності. Після обробки документ відправляється клієнту для завантаження, а тимчасові файли видаляються. Якщо виникає помилка, повертається відповідь з кодом 500 та описом помилки. Цей метод дозволяє перевіряти документ та додає корисні коментарі для редагування.

Метод AddCommentToDocument з контролера:

```
private static void AddCommentToDocument(string inputPath, string outputPath, bool atomicityCheck, bool ambiguityCheck)
 {
    using var wordDocument = WordprocessingDocument.Open(inputPath, false);
    using var outputDocument = (WordprocessingDocument)wordDocument.Clone(outputPath, true);

    var mainPart = outputDocument.MainDocumentPart;
    if (mainPart == null) return;

var commentsPart = mainPart.WordprocessingCommentsPart ?? mainPart.AddNewPart<WordprocessingCommentsPart>();
    commentsPart.Comments ??= new Comments();

    var comments = commentsPart.Comments;
    int commentId = comments.Count() + 1;

    if (atomicityCheck)
    {
       CheckAtomicity(comments, ref commentId, mainPart);
    }

    if (ambiguityCheck)
    {
       CheckAmbiguity(comments, ref commentId, mainPart);
    }

    comments.Save();
    mainPart.Document.Save();
 }
}
```

Цей метод обробляє сам документ, додаючи коментарі до відповідних частин тексту. Він відкриває документ і перевіряє, чи є частина для коментарів. Якщо її немає, створюється нова частина для коментарів. Далі, в залежності від вибору користувача, викликаються методи для перевірки атомарності (CheckAtomicity) та двозначності (CheckAmbiguity). Якщо перевірка атомарності активна, метод додає відповідні коментарі для кожного порушення атомарності. Якщо активна перевірка на двозначність, додаються коментарі для цих випадків. Після завершення коментування зберігаються зміни в документі.

Метод CheckAtomicity з контролера:

```
    private static void CheckAtomicity(Comments comments, ref int commentId, MainDocumentPart mainPart)
    {
        string author = "Атомарність";
        var body = mainPart.Document.Body;
        if (body == null) return;
        var paragraphs = body.Elements<Paragraph>().ToList();


        int wordCountLimit = 15;

        foreach (var paragraph in paragraphs)
        {
            var text = paragraph.InnerText;

            if (string.IsNullOrWhiteSpace(text)) continue;

            var sentences = Regex.Split(text, @"(?<=[.!?])\s+");

            foreach (var sentence in sentences)
            {
                var wordCount = sentence.Split(new[] { ' ', '\t', '\n' }, StringSplitOptions.RemoveEmptyEntries).Length;

                if (wordCount > wordCountLimit)
                {
                    AddCommentToText(comments, ref commentId,
                        $"Речення містить більше {wordCountLimit} слів і може бути занадто складним для атомарності.",
                        author, mainPart, sentence);
                }
            }
        }
    }
}
```

Цей метод перевіряє атомарність документу, розділяючи текст на речення і перевіряючи кількість слів у кожному. Якщо речення містить більше 15 слів, воно вважається надто складним для атомарності, і додається коментар.

Метод CheckAmbiguity з контролера:

```
    private static void CheckAmbiguity(Comments comments, ref int commentId, MainDocumentPart mainPart)
    {
        string author = "Двозначність";
        var body = mainPart.Document.Body;
        if (body == null) return;
        var text = body.InnerText;


        var ambiguousTerms = FindAmbiguousTerms(text);

        foreach (var term in ambiguousTerms)
        {
            AddCommentToText(comments, ref commentId,
```

```
                $"Терміну '{term}' - ймовірно є двозначністю. Переконайтесь, що додали уточнення щодо його значення.",
                author, mainPart, term);
        }
    }
```

Цей метод перевіряє наявність двозначних термінів у документі. Для цього він аналізує весь текст документа, шукаючи терміни, що можуть бути неоднозначними. Якщо знайдено терміни, такі як "швидк", "зручн", "легк" та інші, додається коментар до кожного з таких термінів, вказуючи, що цей термін може бути двозначним і потребує уточнення.

Метод FindAmbiguousTerms з контролера:

```
    private static List<string> FindAmbiguousTerms(string text)
    {
        var ambiguousTerms = new List<string>
        {
            "швидк", "зручн", "легк", "прост", "ефективн", "оптималь", "велик", "мал", "сучасн"
        };

        var foundTerms = new List<string>();

        foreach (var term in ambiguousTerms)
        {
            var pattern = $@"\b{Regex.Escape(term)}\w*\b";
            var matches = Regex.Matches(text, pattern, RegexOptions.IgnoreCase);

            foreach (Match match in matches)
            {
                foundTerms.Add(match.Value);
            }
        }

        return foundTerms.Distinct().ToList();
    }
```

Цей метод знаходить усі терміни, що відповідають заданому списку потенційно двозначних слів. Він використовує регулярні вирази для пошуку в тексті документу всіх варіантів слів, що починаються з вказаних коренів (наприклад, "швидк", "легк" і т.д.). Після знаходження таких термінів метод повертає список усіх унікальних термінів.

Метод AddCommentToText з контролера:

```
    private static void AddCommentToText(Comments comments, ref int commentId, string text, string author, MainDocumentPart mainPart, string term)
    {
```

```
    var body = mainPart.Document.Body;
    if (body == null) return;
    var runs = body.Descendants<Run>().ToList();


    foreach (var run in runs)
    {
        var runText = run.InnerText;

        if (runText.Contains(term, StringComparison.OrdinalIgnoreCase))
        {
            var comment = new Comment()
            {
                Id = commentId.ToString(),
                Author = author,
                Date = DateTime.Now
            };

            comment.AppendChild(new Paragraph(new Run(new Text(text))));
            comments.AppendChild(comment);

            run.InsertBeforeSelf(new CommentRangeStart() { Id = commentId.ToString() });
            run.AppendChild(new CommentRangeEnd() { Id = commentId.ToString() });
            run.AppendChild(new Run(new CommentReference() { Id = commentId.ToString() }));

            commentId++;
        }
    }
}
```

Цей метод додає коментарі до тексту в документі Word. Він шукає всі місця в тексті, де зустрічається певне слово або термін, і додає до цих місць коментар. Коментар пояснює, чому цей термін може бути проблемним, наприклад, через двозначність. Метод використовує спеціальну структуру коментарів у Word, щоб вставити посилання на коментар безпосередньо до тих частин тексту, де знайдено термін. Це дозволяє кожному коментарю бути прив'язаним до конкретного фрагмента тексту в документі, щоб користувач міг побачити, до чого саме він стосується

Метод Atomicity та Explanation з контролера:

```
    public IActionResult Atomicity()
    {
        return View();
    }

    public IActionResult Explanation()
    {
        return View();
    }
```

Ці методи є простими екшенами контролера, які відповідають за завантаження відповідних сторінок (представлень) у веб-додатку. Вони

просто повертають представлення, в якому користувач може взаємодіяти з функціями перевірки атомарності та пояснень щодо перевірки документів. Вони не здійснюють обробку даних, а лише надають доступ до відповідних інтерфейсів.

Клас DocumentModel:

```
using System.ComponentModel.DataAnnotations;

namespace Atomicity.Models
{
   public class DocumentModel
   {
      [Required]
      public string FileName { get; init; }
      public long FileSize { get; set; }
      public string FilePath { get; init; }

      public DocumentModel(string fileName, string filePath) =>
         (FileName, FilePath) = (fileName, filePath);
   }
}
```

Цей клас використовується для збереження метаданих документа, таких як ім'я файлу, розмір файлу та шлях до файлу. Він є моделлю для роботи з документами в додатку.

Загальна структура _Layout.cshtml:

```
@model Atomicity.Models.DocumentModel

<!DOCTYPE html>
<html lang="uk">
<head>
   <meta charset="utf-8" />
   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
   <title>@ViewData["Title"]</title>
   <link rel="icon" href="/images/favicon.ico" type="image/x-icon" />
   <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
   <link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
</head>
<body>
   <header>
      <nav class="navbar navbar-expand-sm navbar-light bg-white border-bottom box-shadow mb-3">
         <div class="container-fluid d-flex justify-content-between">
            <a class="navbar-brand d-flex align-items-center" asp-area="" asp-controller="Pages" asp-action="Atomicity">
               <span>Атомарність і двозначність</span>
            </a>

            <div class="d-flex me-auto">
               <div class="dropdown">
```

```
                        <a class="btn btn-light dropdown-toggle" href="#" role="button" id="dropdownMenuLink"
data-bs-toggle="dropdown" aria-expanded="false">
                    Перевірятu
                </a>
                <ul class="dropdown-menu" aria-labelledby="dropdownMenuLink">
                    <li>
                        <div class="form-check align-items-center d-flex gap-2">
                    <input class="form-check-input" type="checkbox" id="atomicityCheck" checked style="margin-left: -0.8rem;">
                            <label class="form-check-label" for="atomicityCheck">
                                Атомарність
                            </label>
                        </div>
                    </li>
                    <li>
                        <div class="form-check align-items-center d-flex gap-2">
                <input class="form-check-input" type="checkbox" id="ambiguityCheck" checked style="margin-left: -0.8rem;">
                            <label class="form-check-label" for="ambiguityCheck">
                                Двозначність
                            </label>
                        </div>
                    </li>
                </ul>
            </div>
        </div>

        <ul class="navbar-nav">
            <li class="nav-item">
          <a class="nav-link" asp-area="" asp-controller="Pages" asp-action="Explanation">Пояснення функціоналу</a>
            </li>
        </ul>
    </div>
</nav>
</header>

<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2025 - Atomicity and ambiguity
    </div>
</footer>

<script src="~/lib/jquery/dist/jquery.min.js"></script>
<script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
<script src="~/js/site.js" asp-append-version="true"></script>
</body>
</html>
```

Файл _Layout.cshtml є загальним шаблоном для веб-сторінок і визначає структуру сайту, включаючи мета-дані, заголовок, меню навігації, основний контент та футер. У ньому налаштовуються стилі та скрипти, що використовуються на всіх сторінках, а також визначається місце для вставки специфічного контенту через конструкцію @RenderBody(), що дозволяє кожній сторінці наслідувати цей макет і зберігати єдину структуру на всьому сайті.

Сторінка Atomicity.cshtml:

```
    @model Atomicity.Models.DocumentModel

@{
    ViewData["Title"] = "Атомарність і двохзначність";
}

<div class="file-upload-container">
    <div class="file-upload-box">
        <div class="left-box">
            <div class="drag-area" id="drag-area">
                <img src="/images/drag_drop.png" alt="Перетягніть файл" class="drag-image" />
                <p id="drag-text">Перетягніть файл</p>
            </div>
        </div>

        <div class="right-box">
            <h2>Або виберіть файл</h2>
            <form enctype="multipart/form-data" method="post" asp-action="UploadDocument">
                <input type="file" name="file" id="file" accept=".docx" style="display: none;" />
                <button type="button" class="upload-btn" id="choose-file-btn">Переглянути файли</button>
            </form>

            <div class="file-info">
                <p>Підтримувані типи файлів: .docx</p>
            </div>
        </div>
    </div>

    <div class="file-preview-container" id="file-preview-container" style="display: none;">
        <div class="file-preview" id="file-preview">
            <div class="file-details">
                <p id="file-name"></p>
                <p id="file-size"></p>
            </div>
            <button id="remove-file" class="remove-btn">×</button>
        </div>

        <div class="action-btn-container">
            <button id="action-btn" class="upload-btn">Перевірити</button>
        </div>
    </div>
</div>
```

Сторінка Atomicity.cshtml призначена для завантаження документа, вибору параметрів перевірки атомарності та двозначності і запуску процесу перевірки. Вона надає користувачеві можливість завантажити документ через форму або перетягування файлу, переглянути деталі файлу та ініціювати перевірку, а також підтримує формат .docx, що дозволяє аналізувати документи на виявлення проблем з атомарністю та двозначністю.

Сторінка Explanation.cshtml:

```
@model Atomicity.Models.DocumentModel

@{
    ViewData["Title"] = "Пояснення функціоналу";
}

<div class="explanation-container">
    <h1>@ViewData["Title"]</h1>

      <p>Цей інструмент допомагає перевірити атомарність та двозначність у документі, що завантажується. Атомарність та двозначність є різними аспектами якості вимог до програмного забезпечення, і цей інструмент перевіряє їх окремо, надаючи детальні коментарі для кожного з аспектів.</p>

    <h2>Як це працює:</h2>
    <ol>
        <li>Завантажте документ (на даний момент підтримуються файли .docx).</li>
          <li>Натиснувши "Перевіримти", інструмент аналізує документ на вибрані вами властивості (атомарність і/або двозначність) та надає відповідні коментарі для кожного з аспектів.</li>
              <li>У головній панелі є меню, де можна вибрати, які саме властивості перевіряти: атомарність, двозначність, або обидва аспекти разом.</li>
          <li>Після перевірки ви зможете завантажити файл з доданими коментарями, що вказують на виявлені проблеми.</li>
    </ol>

    <h2>Підтримувані формати:</h2>
    <p>На даний момент підтримуються файли формату .docx.</p>

    <h2>Що таке атомарність?</h2>
     <p>Атомарність визначається як властивість вимог, коли кожне твердження є самодостатнім і не може бути поділене на менші частини без втрати завершеності. Перевірка атомарності фокусується на тому, чи можна розділити вимогу на більш дрібні елементи, не втрачаючи її повноти. Програма шукає довгі речення і пропонує вам звернути увагу на них, оскільки вони можуть бути надто складними для одного твердження.</p>

    <h2>Що таке двозначність?</h2>
     <p>Двозначність означає наявність різних можливих тлумачень вимоги, що може призвести до неоднозначних інтерпретацій. Інструмент перевіряє, чи є у тексті формулювання, яке може мати кілька трактувань. Програма шукає підозрілі слова, які можуть бути двозначними, і пропонує перевірити, чи було додано до них уточнення для чіткішого розуміння.</p>

    <a href="@Url.Action("Atomicity", "Pages")" class="btn btn-primary">Повернутися до перевірки документа</a>
</div>
```

Сторінка Explanation.cshtml містить детальні пояснення функціоналу інструменту, зокрема, як працює перевірка атомарності та двозначності у документах. Вона роз'яснює, що таке атомарність та двозначність, їх значення для якості документації, а також надає інструкції по використанню інструменту для завантаження документів і отримання коментарів про знайдені проблеми. Ця сторінка допомагає користувачам краще зрозуміти механізм роботи інструменту та його застосування для покращення якості вимог.

Файл site.js:

```
document.addEventListener('DOMContentLoaded', () => {
const dragArea = document.getElementById('drag-area');
const fileInput = document.getElementById('file');
```

```
const chooseFileButton = document.getElementById('choose-file-btn');
const filePreviewContainer = document.getElementById('file-preview-container');
const filePreview = document.getElementById('file-preview');
const fileNameElement = document.getElementById('file-name');
const fileSizeElement = document.getElementById('file-size');
const removeButton = document.getElementById('remove-file');
const actionButton = document.getElementById('action-btn');
const fileUploadBox = document.querySelector('.file-upload-box');

const atomicityCheck = document.getElementById('atomicityCheck');
const ambiguityCheck = document.getElementById('ambiguityCheck');

let isFileChecked = false;
let selectedFile = null;

const allowedFileTypes = ['application/vnd.openxmlformats-officedocument.wordprocessingml.document'];

chooseFileButton.addEventListener('click', () => {
    fileInput.click();
});

fileInput.addEventListener('change', () => {
    const file = fileInput.files[0];
    if (file) {
        if (isFileAllowed(file)) {
            handleFile(file);
        } else {
            alert('Будь ласка, виберіть файл формату .docx');
        }
    }
});

dragArea.addEventListener('dragover', (event) => {
    event.preventDefault();
    dragArea.classList.add('dragging');
});

dragArea.addEventListener('dragleave', () => {
    dragArea.classList.remove('dragging');
});

dragArea.addEventListener('drop', (event) => {
    event.preventDefault();
    dragArea.classList.remove('dragging');
    const file = event.dataTransfer.files[0];
    if (file && isFileAllowed(file)) {
        handleFile(file);
    } else {
        alert('Будь ласка, перетягніть файл формату .docx');
    }
});

function isFileAllowed(file) {
    return allowedFileTypes.includes(file.type);
}

function handleFile(file) {
    selectedFile = file;
    fileNameElement.textContent = `Назва: ${file.name}`;
    fileSizeElement.textContent = `Розмір: ${(file.size / 1024).toFixed(2)} KB`;
    filePreviewContainer.style.display = 'flex';
    fileUploadBox.style.display = 'none';

    actionButton.textContent = 'Перевірити';

    isFileChecked = false;
}

removeButton.addEventListener('click', () => {
    fileInput.value = '';
```

```
    filePreviewContainer.style.display = 'none';
    fileUploadBox.style.display = 'flex';
    actionButton.textContent = 'Перевіритu';
    isFileChecked = false;
});

actionButton.addEventListener('click', () => {
    if (!isFileChecked) {
        checkFile();
        isFileChecked = true;

    } else {
        downloadFile();
    }
});

async function checkFile() {
    actionButton.disabled = true;
    actionButton.textContent = 'Перевірка...';

    await new Promise(resolve => setTimeout(resolve, 2000));

    console.log('Файл перевірено');

    const checkedFile = new File([selectedFile], selectedFile.name.replace(/(\.[\w\d_-]+)$/i, '_checked$1'), {
        type: selectedFile.type
    });

    selectedFile = checkedFile;

    fileNameElement.textContent = `Назва: ${selectedFile.name}`;

    actionButton.disabled = false;
    actionButton.textContent = 'Завантажити';
}

function downloadFile() {
    if (!selectedFile) {
        alert('Будь ласка, виберіть файл для завантаження.');
        return;
    }

    const formData = new FormData();
    formData.append('file', selectedFile);
    formData.append('atomicityCheck', atomicityCheck.checked);
    formData.append('ambiguityCheck', ambiguityCheck.checked);

    fetch('/Pages/UploadDocument', {
        method: 'POST',
        body: formData,
    })
        .then(response => response.blob())
        .then(blob => {
            const url = URL.createObjectURL(blob);
            const a = document.createElement('a');
            a.href = url;
            a.download = selectedFile.name;
            a.click();
            URL.revokeObjectURL(url);
        })
        .catch(error => {
            console.error('Помилка завантаження файлу:', error);
            alert('Сталася помилка при завантаженні файлу.');
        });
}

atomicityCheck.addEventListener('change', () => {
    if (actionButton.textContent === 'Перевірка...') {
        atomicityCheck.checked = !atomicityCheck.checked;
    } else {
```

```
            preventUnchecking(atomicityCheck, ambiguityCheck);
        }
    });

    ambiguityCheck.addEventListener('change', () => {
        if (actionButton.textContent === 'Перевірка...') {
            ambiguityCheck.checked = !ambiguityCheck.checked;
        } else {
            preventUnchecking(ambiguityCheck, atomicityCheck);
        }
    });


    function preventUnchecking(checkbox1, checkbox2) {
        if (!checkbox2.checked) {
            checkbox1.checked = true;
        }
    }

    window.addEventListener('resize', updateLayout);

    function updateLayout() {
        const isLandscape = window.innerWidth > window.innerHeight;
        if (isLandscape) {
            fileUploadBox.style.flexDirection = 'row';
        } else {
            fileUploadBox.style.flexDirection = 'column';
        }
    }
});
```

Основні методи, що містяться в скрипті site.js, відповідають за обробку файлів, їх завантаження, перевірку на атомарність і двозначність, а також за надання можливості завантаження оброблених файлів.

DOMContentLoaded: Цей обробник події гарантує, що весь DOM буде завантажений перед виконанням коду. Всі елементи доступні для маніпуляцій, коли ця подія спрацьовує.

chooseFileButton.addEventListener('click', () => {...}): Кнопка "Обрати файл" відкриває стандартний діалог вибору файлів через клік на елемент fileInput. Це дозволяє користувачеві обрати файл без необхідності перетягувати його.

fileInput.addEventListener('change', () => {...}): Цей обробник події спрацьовує, коли користувач вибирає файл. Якщо файл підходить за типом (формат .docx), його обробляє функція handleFile.

dragArea.addEventListener('dragover', (event) => {...}): При перетягуванні файлу над областю dragArea додатково додається клас dragging для візуальної індикації того, що файл можна відпустити.

dragArea.addEventListener('dragleave', () => {...}): Коли файл перестає бути в області перетягування, клас dragging видаляється.

dragArea.addEventListener('drop', (event) => {...}): Після того, як файл буде скинуто в область перетягування, спрацьовує ця подія. Вона перевіряє файл на відповідність типу і обробляє його за допомогою handleFile.

isFileAllowed(file): Перевіряє тип файлу на відповідність списку дозволених типів (в даному випадку тільки .docx).

handleFile(file): Ця функція отримує файл, відображає його ім'я і розмір на сторінці, приховує контейнер завантаження файлів і показує прев'ю файлу.

removeButton.addEventListener('click', () => {...}): Відповідає за скидання вибраного файлу: очищує інпут, приховує прев'ю файлу і відновлює первісний стан кнопки для повторного вибору.

actionButton.addEventListener('click', () => {...}): Якщо файл ще не перевірено, то викликається перевірка файлу, після чого кнопка змінює текст на "Завантажити". Якщо файл уже перевірено, то запускається завантаження.

checkFile(): Симулює процес перевірки файлу (очікування 2 секунди через setTimeout) і змінює назву файлу на "_checked" в його імені. Після завершення перевірки змінює текст на кнопці на "Завантажити".

downloadFile(): Функція для завантаження перевіреного файлу. Вона відправляє файл через POST на сервер, отримує оброблений файл і автоматично ініціює його завантаження через тег <a>.

atomicityCheck.addEventListener('change', () => {...}) і

ambiguityCheck.addEventListener('change', () => {...}): Відповідають за поведінку чекбоксів. Вони перевіряють чи не заблокована кнопка перевірки, та змінюють статус чекбоксів в залежності від того, чи активна перевірка.

preventUnchecking(checkbox1, checkbox2): Функція запобігає зняттю одного чекбокса, якщо інший не обраний. Це забезпечує, що хоча б один чекбокс завжди буде активним під час перевірки.

window.addEventListener('resize', updateLayout): Слухач події зміни розміру вікна, що допомагає адаптувати макет до зміни орієнтації екрану.

updateLayout(): Функція, яка змінює напрямок розташування елементів в залежності від орієнтації екрана. Якщо ширина екрану більша за висоту, елементи розташовуються в ряд, інакше — в колонку.

Файл Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

var app = builder.Build();

if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Pages}/{action=Atomicity}/{id?}");

app.Run();
```

Файл Program.cs налаштовує веб-додаток ASP.NET Core. Спочатку створюється об'єкт для конфігурації додатку. Потім додається підтримка контролерів та представлень. Якщо додаток не в режимі розробки, включаються налаштування для обробки помилок та забезпечення безпеки з'єднань через HTTPS. Далі активується обробка статичних файлів, маршрутизація запитів до контролерів, та авторизація для перевірки доступу. Визначається основний маршрут за замовчуванням, і додаток запускається для обробки запитів.

3.3. Опис результатів роботи програми.

Для демонстрування результатів роботи розробленого web-додатку було розроблено демо-сайт для перевірки документації IT-проєкту на атомарність та двозначність .

Вигляд головної сторінки даного web-додатку під назвою “Атомарність і двозначність” подано на рис. 3.3.1.

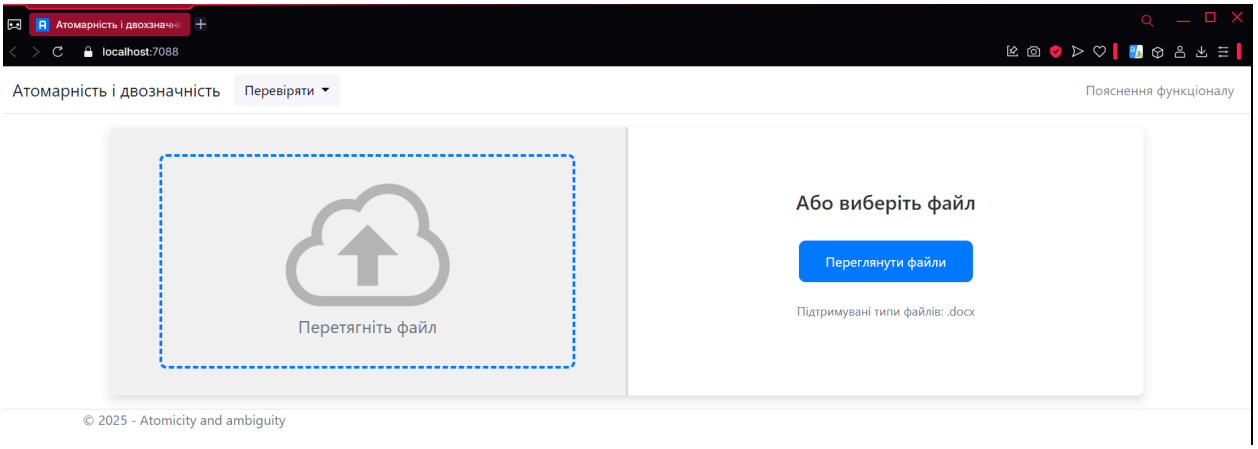

Рис. 3.3.1. Головна сторінка без вибраного файлу

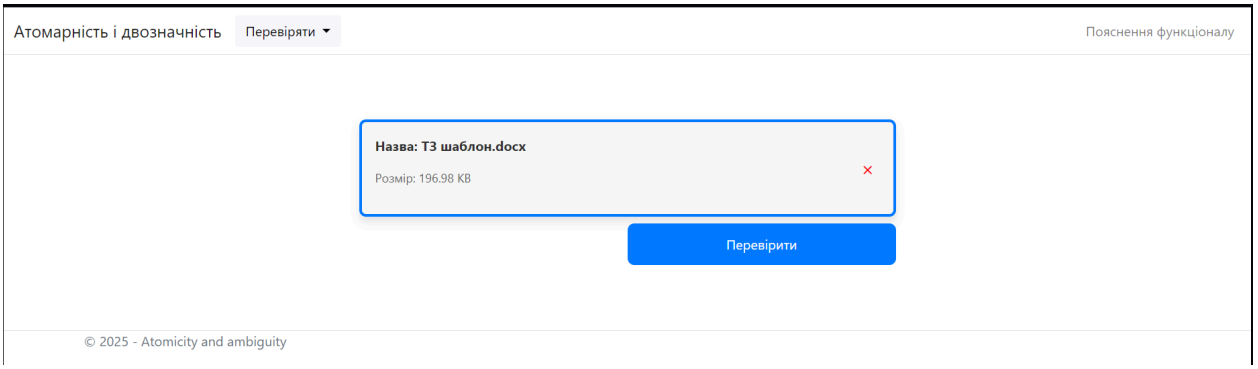

Рис. 3.3.2. Головна сторінка з вибраним файлом

Під час будь-якого етапу можна обирати типи перевірок (атомарність і/або двозначність). Повина бути, як мінімум одна перевірка, програм забороняє знімати останню галочку.

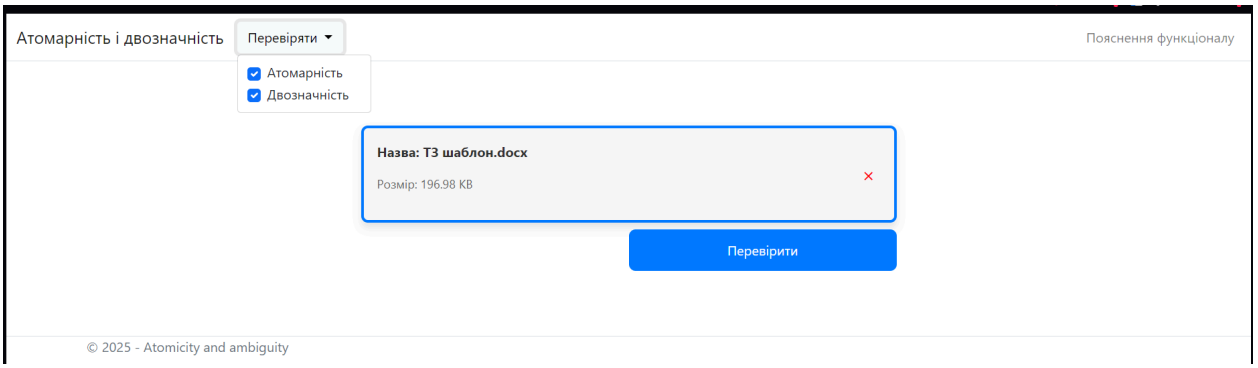

Рис. 3.3.3. Головна сторінка з вибраним файлом і розкритим меню вибору перевірок

Після натискання кнопки “Перевірити”, іде етап “Перевірка…”.

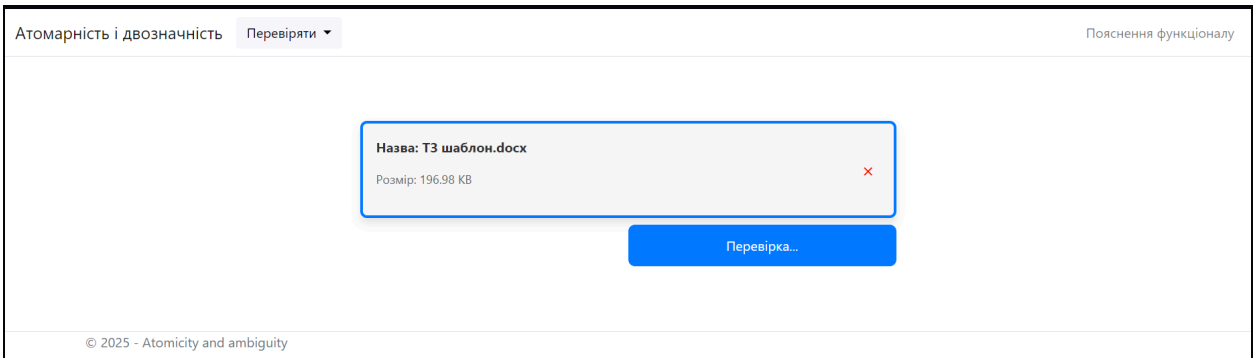

Рис. 3.3.4. Головна сторінка з вибраним файлом на етапі “Перевірка…”

Опісля етапу “Перевірка…” можна завантажити перевірений файл з назвою <ім’я файлу>_checked.docx. Зауважу, що якщо перед завантаженням файлу змінити галочки перевірки завантажений файл міститиме коментарі лише до наявних типів перевірки.

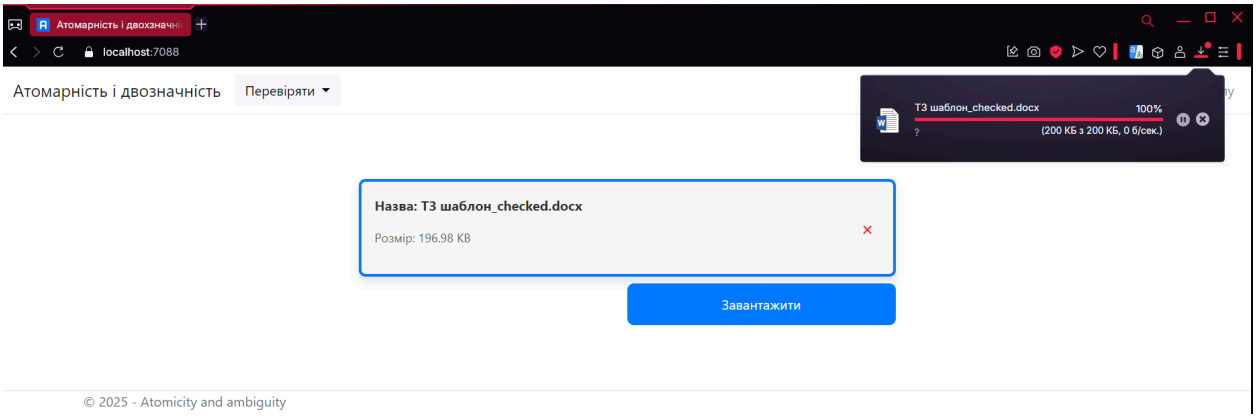

Рис. 3.3.5. Головна сторінка з вибраним файлом на етапі "Завантажити" під час завантаження файлу

Для ознайомлення з програмою є сторінка "Пояснення функціоналу". На неї можна потрапити натиснувши на відповідне посилання на верхній панелі зправа.

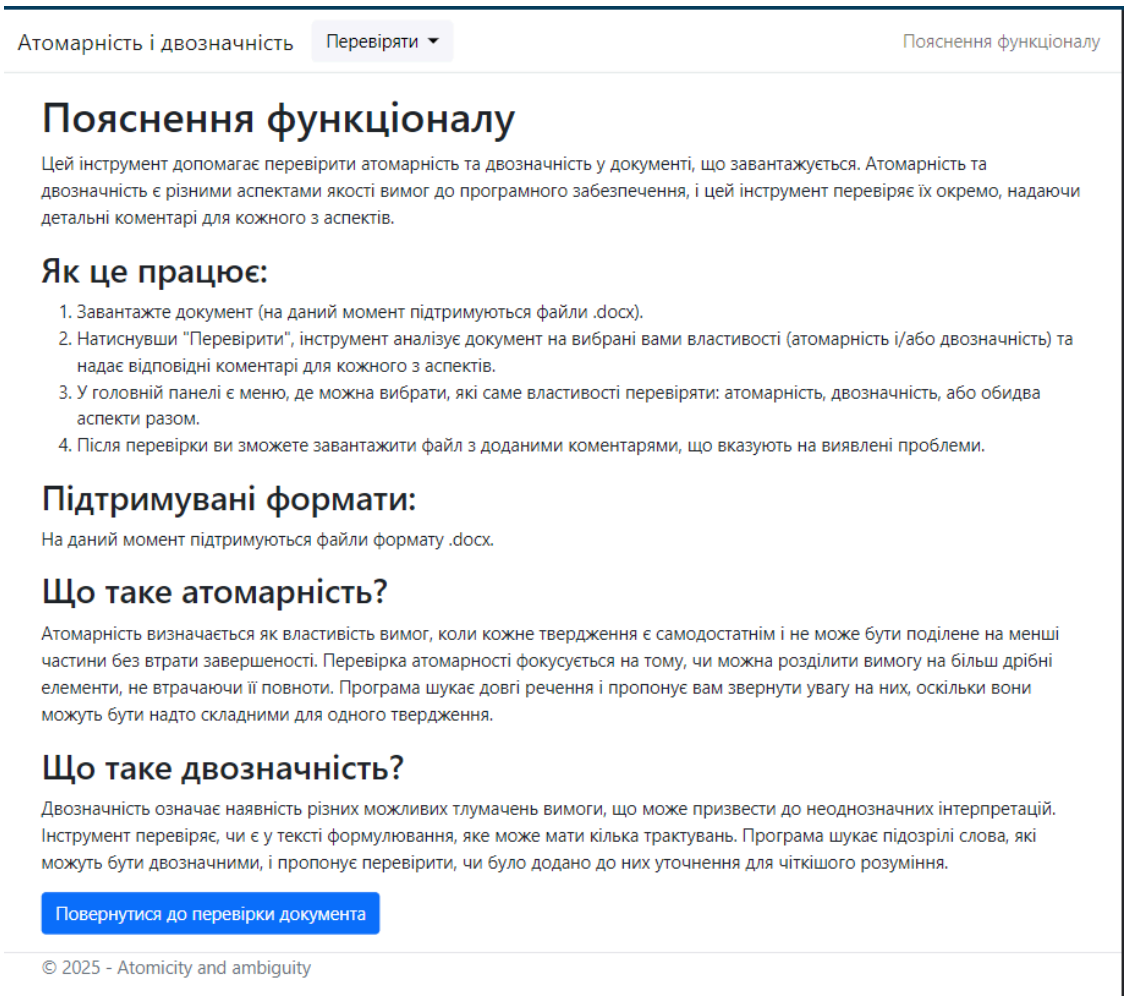

Рис. 3.3.6. Сторінка "Пояснення функціоналу"

Далі зображено частину файлу .docx зі згенерованими коментарями щодо атомарності та двозначності.

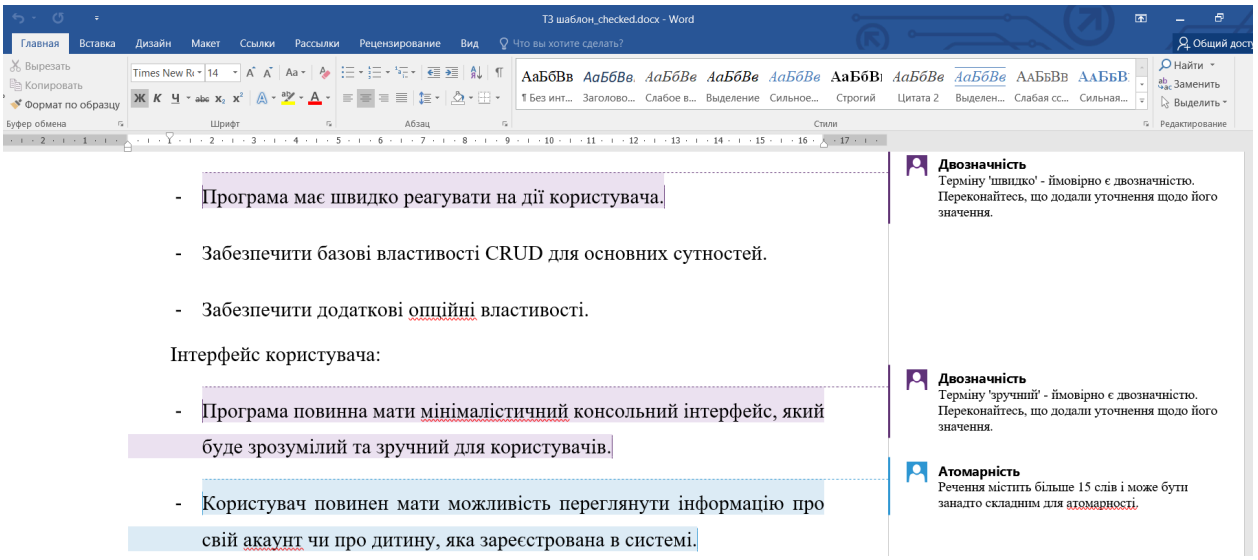

Рис. 3.3.7. Частину перевіреного файлу .docx з коментарями

Як видно, розроблений інструмент допомагає автоматизувати процес перевірки документації на відповідність визначеним критеріям. Програма виділяє найбільш підозрілі місця, дозволяючи користувачам зосередитись на їх аналізі. Остаточне рішення щодо внесення змін залишається за користувачем.

ВИСНОВКИ

Метою даної курсової роботи було розроблення програмного забезпечення для перевірки наданої користувачем документації ІТ-проєкту на атомарність та двозначність. Підставою для розроблення стало завдання на виконання курсової роботи з дисципліни «Компоненти програмної інженерії. Частина 4. Якість та тестування програмного забезпечення» студентами ІІІ курсу кафедри ПЗКС НТУУ «КПІ ім. І.Сікорського».

Для досягнення поставленої мети виконано всі заплановані завдання; розроблено необхідні графічні матеріали; реалізовано всі вимоги до програмного продукту, програмного та апаратного забезпечення; створено відповідну документацію.

Розроблений програмний продукт представляє собою сайт з перевірки наданої користувачем документації ІТ-проєкту на атомарність та двозначність.

Програму створено мовою програмування C#, використовуючи ASP .NET Core WebApplication версії 8.0 з використанням шаблону проєктування MVC.

Перспективним напрямом подальшого дослідження даної тематики є розширення функціональності програми, впровадження додаткових функцій та можливостей, наприклад, удосконалення наявних перевірок, додавання нових типів перевірок або впровадження оброблення нових форматів файлів.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Мартін, Роберт С. "Чистий код: Створення та підтримка добре написаного коду" / Роберт С. Мартін; пер. з англ. - К.: Діалектика, 2019. - 464 с.

2. Microsoft [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://support.microsoft.com/uk-ua/office/формати-open-xml-і-розширення-імен-файлів-5200d93c-3449-4380-8e11-31ef14555b18 (дата звернення: 20.01.2025) - Назва з екрана.

3. QALearning [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://qalearning.com.ua/theory/lectures/material/requirements-testing-methods-equivalence/ (дата звернення: 20.01.2025) - Назва з екрана.

4. Financial LNU [Електронний ресурс]: [Веб-сайт]. - Режим доступу: https://financial.lnu.edu.ua/wp-content/uploads/2019/09/LEKTSIYA-3.pdf (дата звернення: 20.01.2025) - Назва з екрана.