Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет Комп'ютерних технологій та систем

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка
до кваліфікаційної роботи
ОС магістра

на тему «Дослідження методів верифікації вимог до сайтів»

за освітньою програмою **Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав студент групи ПЗ2322: /Нікіта ЛОХВИЦЬКИЙ/

Керівник: /доцент Олена КУРОП'ЯТНИК/

Нормоконтролер: /доцент Світлана ВОЛКОВА/

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2025

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems

Department Computer information technology

Explanatory Note
to Master's Thesis

on the topic: «Research of methods of verification of requirements for sites»

according to educational curriculum **software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2322: /Nikita LOKHVITSKY/

Scientific Supervisor: /Olena KUROPIATNYK/

Normative controller : /Svitlana VOLKOVA/

Dnipro – 2025

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»
Кафедра: «Комп’ютерні інформаційні технології»
Рівень вищої освіти: магістр
Освітня програма: «Інженерія програмного забезпечення»
Спеціальність: «121 Інженерія програмного забезпечення»
(шифр та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри КІТ
_____ Вадим ГОРЯЧКІН
(підпис) (Ім’я ПРІЗВИЩЕ)

Дата _____

З А В Д А Н Н Я

на кваліфікаційну роботу для здобуття ступеня магістра
студенту Лохвицькому Нікіті Сергійовичу
1. Тема роботи: «Дослідження методів верифікації вимог до сайтів»
Керівник роботи: Куроп'ятник Олена Сергіївна
затверджені наказом 1187 ст від 29.12.2023р.__
2. Строк подання студентом роботи: 10.01.2025р.
3. Вихідні дані до роботи: _____
4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати)
4.1 Аналіз сучасних засобів верифікації вимог до сайтів
4.2 Методика та дослідження
4.3 Проектування й розробка
4.4 Дослідження ефективності програми
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація, відео роботи програми

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Вступ	03.03.24-14.04.24	
2	Аналіз сучасного стану дослідження проблеми за науковими та літературними джерелами	15.04.24-22.08.24	10%
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	23.08.24-03.11.24	
4	Постановка задачі, технічне завдання	04.11.24-10.11.24	30%
5	Розробка інструментальних засобів дослідження	11.11.24-08.12.24	
6	Виконання досліджень	09.12.24-15.12.24	60%
7	Оформлення пояснювальної записки	16.12.24-05.01.25	
8	Розробка демонстраційних матеріалів	06.01.25-12.01.25	100%
9	Подання кваліфікаційної роботи до кафедри	10.01.25	
10	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	24.01.25	

Студент _____ Нікіта ЛОХВИЦЬКИЙ
(підпис) (Ім’я ПРІЗВИЩЕ)

Керівник роботи _____ доц. Олена КУРОП’ЯТНИК
(підпис) (Ім’я ПРІЗВИЩЕ)

РЕФЕРАТ

Опис магістерської дисертації складається з 76 сторінок, 21 рисунку, 7 таблиць, 4 додатки та 17 джерел.

Предмет дослідження: методи верифікації вимог до веб-сайтів.

Об'єкт дослідження: процес розробки та перевірки вимог до веб-сайтів.

Мета роботи: дослідження сучасних методів верифікації вимог до веб-сайтів, їх аналіз, розробка програмного забезпечення для автоматизації процесу верифікації, а також тестування його ефективності.

У вступі розглянуто актуальність теми, визначено мету, завдання та обґрунтовано актуальність теми.

У першому розділі проаналізовано сучасні методи та інструменти верифікації вимог. Розглянуто ручні, автоматизовані та комбіновані підходи, а також існуючі проблеми, такі як нечіткість вимог і труднощі автоматизації.

У другому розділі проведено дослідження ефективності обраних методів верифікації. Розроблено критерії оцінки, обрано показники та методи для аналізу текстів вимог. Досліджено семантичні помилки, нечіткі формулювання, неоднозначності та стилістичні помилки.

У третьому розділі описано процес проектування й розробки програмного забезпечення. Проведено зовнішнє проектування з визначенням функціональних вимог, вхідних і вихідних даних. Реалізовано внутрішнє проектування із застосуванням алгоритмів обробки тексту.

У четвертому розділі проведено тестування та налагодження програмного забезпечення, що дозволило оцінити його ефективність.

Ключові слова: ВЕРИФІКАЦІЯ, ВИМОГИ ДО САЙТІВ, PYTHON, АВТОМАТИЗАЦІЯ, АНАЛІЗ ТЕКСТУ, СПРОЩЕННЯ РОЗРОБКИ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ

NLP (Natural Language Processing) – обробка природної мови. Галузь штучного інтелекту, що займається аналізом та розумінням людської мови.

SPACY – бібліотека Python для обробки природної мови, що надає широкий спектр функцій для аналізу тексту.

GPT (Generative Pre-trained Transformer) — генеративна модель попередньо навченого трансформера. Потужна мовна модель від OpenAI.

NLTK (Natural Language Toolkit) — набір інструментів природної мови. Бібліотека Python для роботи з текстами.

UI (User Interface) – інтерфейс користувача. Засіб взаємодії користувача з програмою.

GUI (Graphical User Interface) – графічний інтерфейс користувача. Тип інтерфейсу, що використовує графічні елементи для взаємодії з користувачем.

QA (Quality Assurance) – забезпечення якості. Комплекс заходів, спрямованих на забезпечення якості програмного забезпечення.

TXT – текстовий файл. Файл, що містить неформатований текст.

DOCX – документ Microsoft Word (формат Office Open XML).

PDF (Portable Document Format) – портативний формат документів. Формат файлу для представлення документів у незалежному від програмного забезпечення, апаратного забезпечення та операційної системи вигляді.

ВСТУП

Актуальність дослідження. У сучасному світі веб-технології відіграють ключову роль у різних сферах життя, від бізнесу та освіти до розваг і соціальних комунікацій. Кількість веб-сайтів стрімко зростає, що призводить до підвищення вимог до їхньої якості. Нечіткі або некоректні вимоги до сайтів можуть призвести до серйозних помилок у розробці, додаткових витрат і втрати часу. Тому автоматизація процесу перевірки вимог до веб-сайтів стає важливим завданням для підвищення якості розробки програмного забезпечення.

На сьогоднішній день існує велика кількість інструментів та підходів для аналізу тексту і верифікації вимог, проте більшість із них або мають обмежену функціональність, або вимагають значних ресурсів для впровадження. Зокрема, ручні методи перевірки залишаються трудомісткими й суб'єктивними, тоді як автоматизовані рішення ще не досягли достатньої гнучкості та універсальності для застосування в різних проєктах.

Актуальність теми також обумовлена зростаючим попитом на якісне програмне забезпечення і потребою у створенні інструментів, здатних виявляти помилки, неоднозначності та неточності в текстах вимог. Це дозволяє скоротити час на виправлення помилок і зменшити ризики під час реалізації проєктів.

Метою кваліфікаційної роботи є дослідження існуючих методів верифікації вимог до веб-сайтів, розробка програмного забезпечення для автоматизації верифікації вимог до веб-сайтів.

Об'єктом дослідження є процес формування і верифікації вимог до веб-сайтів. Предметом дослідження є методи і технології, що використовуються для автоматизації цього процесу.

Дослідження спрямоване на розв'язання наступних завдань: аналіз сучасних інструментів і методів верифікації, розробка алгоритмів для

автоматизованого аналізу текстів, створення програмного продукту, що інтегрує ці алгоритми, та оцінка ефективності розробленого рішення за допомогою експериментального тестування.

Для досягнення поставленої мети в роботі застосовано методи системного аналізу, порівняльного аналізу існуючих підходів до верифікації вимог, методи обробки природної мови (NLP), зокрема використання LanguageTool API, SpaCy та NLTK, а також методи розробки програмного забезпечення.

Наукова новизна роботи полягає у тому, що було запропоновано показники верифікації вимог, розроблено алгоритмічне забезпечення для автоматизації верифікації вимог.

Практичне значення роботи полягає у створенні інструменту, який може бути використаний розробниками веб-сайтів для підвищення якості вимог, скорочення часу на їх верифікацію та зменшення ризиків, пов'язаних з помилками у специфікаціях веб-проектів.

Результати заслухано на наукових семінарах кафедри «Комп’ютерні інформаційні технологій» 14.06.2024 та 10.01.2025

РОЗДІЛ 1. АНАЛІЗ СУЧАСНИХ ЗАСОБІВ ВЕРИФІКАЦІЇ ВИМОГ ДО САЙТІВ

У сучасному цифровому світі веб-сайти стали невід'ємною частиною бізнесу, комунікації, освіти та багатьох інших сфер життя. Вони є потужним інструментом для представлення інформації, надання послуг, здійснення комерційної діяльності та взаємодії з користувачами. Зі зростанням складності веб-проектів та збільшенням вимог до їхньої функціональності, надійності та безпеки, питання забезпечення якості розробки веб-сайтів набуває особливої актуальності. Одним із ключових факторів успіху будь-якого веб-проекту є наявність чітко сформульованих та верифікованих вимог.

Вимоги до веб-сайту визначають його функціональність, поведінку, продуктивність, безпеку, зручність використання та інші важливі характеристики. Вони слугують основою для проектування, розробки, тестування та подальшої підтримки веб-ресурсу. Проте, навіть найретельніше сформульовані вимоги потребують ретельної перевірки та підтвердження їхньої коректності, повноти, несуперечливості та відповідності потребам замовника та кінцевих користувачів. Цей процес, відомий як верифікація вимог, є критично важливим для забезпечення успішної реалізації веб-проекту.

Верифікація вимог є невід'ємною частиною життєвого циклу розробки програмного забезпечення, зокрема веб-сайтів. Її метою є виявлення та усунення помилок, неточностей та суперечностей у вимогах на ранніх етапах розробки, що дозволяє запобігти значним витратам часу, ресурсів та фінансів у майбутньому. Нечіткі, неоднозначні, неповні або суперечливі вимоги можуть призвести до неправильної інтерпретації, помилок у проектуванні та розробці, незадоволення замовника, низької якості кінцевого продукту та навіть до повного провалу проекту.

У зв'язку з цим, дослідження та аналіз сучасних методів та засобів верифікації вимог до веб-сайтів є надзвичайно актуальним та важливим завданням. У даному розділі буде проведено детальний огляд існуючих підходів до верифікації, розглянуто їхні переваги та недоліки, а також визначено основні проблеми та тенденції у цій області. Зокрема, в цьому розділі буде розглянуто:

а) загальні поняття верифікації вимог, включаючи визначення, значення, типи вимог та етапи процесу верифікації;

б) різноманітні методи верифікації вимог, як ручні (перегляд, інтерв'ю, прототипування), так і автоматизовані (аналіз специфікацій, формальні методи);

в) існуючі інструменти та засоби, що використовуються для верифікації вимог, такі як інструменти для управління вимогами, моделювання та аналізу, автоматизованого тестування;

г) існуючі проблеми та обмеження сучасних методів верифікації, включаючи труднощі збору вимог, складність автоматизації та людський фактор.

Аналіз сучасних засобів верифікації вимог, проведений у цьому розділі, слугуватиме теоретичною основою для подальшого дослідження та розробки власної методики верифікації, яка буде представлена в наступних розділах даної роботи. Результати цього аналізу дозволять визначити найбільш ефективні підходи та інструменти для забезпечення якості та надійності веб-сайтів на етапі формування вимог.

1.1. Загальні поняття верифікації вимог

Верифікація вимог є критично важливим етапом у процесі розробки будь-якої програмної системи, включаючи веб-сайти. Вона являє собою комплекс заходів, спрямованих на перевірку та підтвердження того, що сформульовані вимоги є коректними, повними, несуперечливими, однозначними та

відповідають потребам замовника та кінцевих користувачів. Іншими словами, верифікація вимог відповідає на питання: "Чи правильно ми сформулювали вимоги?".

Значення верифікації вимог важко переоцінити. Помилки, допущені на етапі формування вимог, можуть мати серйозні наслідки для всього проекту. Нечіткі, неоднозначні або суперечливі вимоги призводять до неправильної інтерпретації, помилок у проектуванні, розробці, тестуванні та, як наслідок, до незадоволення замовника та низької якості кінцевого продукту. У гіршому випадку, такі помилки можуть призвести до повного провалу проекту та значних фінансових втрат.

Верифікація вимог є проактивним процесом, який спрямований на попередження помилок, а не на їх виправлення після того, як вони вже були допущені. Вона дозволяє виявити та усунути проблеми на ранніх етапах розробки, що значно зменшує вартість та час, необхідні для їх виправлення. За статистикою, вартість виправлення помилок, виявлених на етапі формування вимог, значно нижча, ніж вартість виправлення тих же помилок, виявлених на етапах кодування, тестування або вже після введення системи в експлуатацію.

Процес верифікації вимог тісно пов'язаний з іншими процесами життєвого циклу розробки програмного забезпечення, такими як збір вимог, аналіз вимог, проектування, розробка, тестування та супровід. Він є невід'ємною частиною забезпечення якості програмного забезпечення та управління ризиками проекту.

Верифікація вимог передбачає використання різноманітних методів та технік, які дозволяють перевірити вимоги з різних точок зору. Це можуть бути ручні методи, такі як перегляд документації, інтерв'ю з зацікавленими сторонами, створення прототипів, а також автоматизовані методи, такі як

аналіз специфікацій за допомогою спеціалізованого програмного забезпечення, використання формальних методів верифікації.

У контексті розробки веб-сайтів, верифікація вимог має свої особливості, пов'язані зі специфікою веб-технологій та вимогами до веб-ресурсів. Веб-сайти часто мають складну структуру, інтегруються з різними зовнішніми системами, повинні відповідати вимогам щодо продуктивності, безпеки, зручності використання та доступності. Тому, процес верифікації вимог до веб-сайтів повинен враховувати ці особливості та використовувати відповідні методи та інструменти.

Наступні підпункти цього розділу детальніше розглянуть типи вимог до веб-сайтів та етапи процесу верифікації, що дозволить отримати повне уявлення про загальні поняття верифікації вимог та створити основу для подальшого аналізу сучасних методів та засобів верифікації.

1.1.1. Визначення та значення верифікації вимог

Верифікація вимог – це процес оцінки та перевірки того, чи відповідають сформульовані вимоги до програмного забезпечення, зокрема веб-сайту, заданим критеріям якості, таким як коректність, повнота, несуперечливість, однозначність, зрозумілість, верифікованість та простежуваність. Простіше кажучи, верифікація вимог відповідає на питання: "Чи правильно ми сформулювали вимоги?". Вона фокусується на внутрішній консистентності та якості самих вимог, незалежно від того, чи відповідає створений продукт очікуванням користувачів (це завдання валідації).

Суть верифікації вимог полягає в тому, щоб виявити та усунути помилки, неточності, неоднозначності та суперечності у вимогах на якомога ранніх етапах розробки. Це дозволяє запобігти значним витратам часу, ресурсів та коштів у майбутньому, оскільки виправлення помилок у вимогах набагато дешевше та швидше, ніж виправлення тих же помилок, виявлених на етапах кодування, тестування або вже після введення системи в експлуатацію.

Верифікація вимог відіграє критично важливу роль у забезпеченні успіху веб-проекту з наступних причин:

а) запобігання помилкам та дефектам. Ретельна верифікація дозволяє виявити потенційні проблеми у вимогах ще до початку розробки, що значно зменшує ймовірність появи помилок у коді та, як наслідок, покращує якість кінцевого продукту;

б) зниження витрат та термінів розробки. Виправлення помилок на ранніх етапах розробки є значно дешевшим та швидшим, ніж виправлення тих же помилок на пізніх етапах або вже після запуску веб-сайту. Верифікація дозволяє оптимізувати процес розробки та уникнути зайвих витрат;

в) підвищення задоволеності замовника. Чіткі та верифіковані вимоги забезпечують краще розуміння потреб замовника та дозволяють створити веб-сайт, який повністю відповідає його очікуванням. Це сприяє підвищенню задоволеності замовника та зміцненню ділових відносин;

г) покращення комунікації між зацікавленими сторонами. Процес верифікації передбачає активну взаємодію між замовниками, розробниками, тестувальниками та іншими зацікавленими сторонами. Це сприяє покращенню комунікації, порозуміння та співпраці між ними;

д) забезпечення відповідності стандартам та нормам. Верифікація дозволяє переконатися, що веб-сайт відповідає всім необхідним стандартам та нормам, таким як стандарти веб-доступності, безпеки та продуктивності.

У свою чергу, нечіткі, неоднозначні, неповні або суперечливі вимоги можуть призвести до серйозних негативних наслідків для веб-проекту:

а) неправильна інтерпретація вимог розробниками. Різні члени команди розробників можуть по-різному зрозуміти нечіткі вимоги, що призведе до створення різних частин веб-сайту, які не будуть узгоджуватися між собою;

б) помилки у проектуванні та розробці. Неповні або суперечливі вимоги можуть призвести до помилок у проектуванні архітектури веб-сайту, бази даних, інтерфейсу користувача та інших компонентів;

в) затримки у термінах виконання проекту. Виправлення помилок, спричинених нечіткими вимогами, потребує додаткового часу та ресурсів, що може призвести до затримки у термінах виконання проекту;

г) збільшення вартості розробки. Виправлення помилок на пізніх етапах розробки є значно дорожчим, ніж їх попередження на етапі верифікації вимог;

д) незадоволеність замовника та кінцевих користувачів. Веб-сайт, розроблений на основі нечітких або суперечливих вимог, навряд чи задовольнить потреби замовника та кінцевих користувачів, що призведе до їх незадоволення;

е) пошкодження репутації компанії-розробника. Низька якість веб-сайту, спричинена проблемами з вимогами, може негативно вплинути на репутацію компанії-розробника.

Таким чином, верифікація вимог є критично важливим процесом, який забезпечує якість, ефективність та успіх розробки веб-сайтів. Вона дозволяє запобігти багатьом проблемам та негативним наслідкам, пов'язаним з нечіткими або суперечливими вимогами.

1.1.2. Типи вимог до веб-сайтів

Вимоги до веб-сайтів можна класифікувати за різними критеріями. Найбільш поширена класифікація поділяє їх на чотири основні категорії: функціональні, нефункціональні, бізнес-вимоги та користувацькі вимоги. Кожна з цих категорій відіграє важливу роль у визначенні характеристик та поведінки веб-сайту.

1.1.2.1. Функціональні вимоги

Функціональні вимоги описують конкретні функції та операції, які повинна виконувати веб-система. Вони визначають, що саме система повинна робити, які вхідні дані вона приймає, які результати вона генерує та які дії користувач може виконувати. Функціональні вимоги є основою для розробки функціональності веб-сайту та тестування його відповідності заданим критеріям. Приклади функціональних вимог:

а) користувач повинен мати можливість зареєструватися на сайті, використовуючи адресу електронної пошти та пароль;

б) система повинна генерувати звіт про продажі за певний період часу;

в) веб-сайт повинен підтримувати пошук товарів за ключовими словами, категоріями та ціновим діапазоном;

г) користувач повинен мати можливість додавати товари до кошика та оформлювати замовлення онлайн;

д) система повинна автоматично надсилати підтвердження замовлення на електронну пошту користувача.

1.1.2.2. Нефункціональні вимоги

Нефункціональні вимоги описують, як система повинна працювати, а не що вона повинна робити. Вони визначають якісні характеристики системи, такі як продуктивність, безпека, зручність використання, надійність, масштабованість, доступність та інші. Нефункціональні вимоги є важливими для забезпечення задоволення користувачів та успішної роботи веб-сайту. Приклади нефункціональних вимог:

а) продуктивність — веб-сторінка повинна завантажуватися не більше ніж за 3 секунди;

б) безпека — система повинна забезпечувати захист від SQL-ін'єкцій та інших видів кібератак;

в) зручність використання — інтерфейс веб-сайту повинен бути інтуїтивно зрозумілим та легким у використанні для користувачів з різним рівнем технічної підготовки;

г) надійність — система повинна працювати без збоїв протягом 99.9% часу;

д) масштабованість — веб-сайт повинен витримувати одночасне відвідування 10 000 користувачів без втрати продуктивності;

е) доступність — веб-сайт повинен бути доступним для користувачів з обмеженими можливостями, відповідно до стандартів WCAG.

1.1.2.3. Бізнес-вимоги

Бізнес-вимоги описують цілі та завдання, які бізнес хоче досягти за допомогою веб-сайту. Вони визначають стратегічні напрямки розвитку веб-проекту та його внесок у досягнення загальних бізнес-цілей. Бізнес-вимоги зазвичай формулюються на високому рівні абстракції та слугують основою для визначення функціональних та нефункціональних вимог. Приклади бізнес-вимог:

а) збільшити обсяг онлайн-продажів на 20% протягом наступного року;

б) підвищити впізнаваність бренду серед цільової аудиторії;

в) залучити нових клієнтів через веб-сайт;

г) покращити рівень обслуговування клієнтів шляхом надання онлайн-підтримки;

д) автоматизувати процес обробки замовлень та скоротити час їх виконання.

1.1.2.4. Користувацькі вимоги

Користувацькі вимоги описують потреби та очікування кінцевих користувачів веб-сайту. Вони визначають, як користувачі будуть використовувати веб-сайт, які завдання вони хочуть вирішити за його допомогою та які враження вони хочуть отримати від взаємодії з ним. Користувацькі вимоги є важливими для забезпечення зручності використання та задоволення потреб користувачів. Приклади користувацьких вимог:

а) користувач хоче швидко та легко знайти потрібний товар на сайті;

б) користувач хоче мати можливість порівнювати різні товари за характеристиками та ціною;

в) користувач хоче отримати детальну інформацію про товар, включаючи фотографії, опис та відгуки інших покупців;

г) користувач хоче мати можливість оплатити замовлення зручним для нього способом;

д) користувач хоче отримувати своєчасну інформацію про статус свого замовлення.

Розуміння різних типів вимог та їх взаємозв'язку є важливим для успішної верифікації та розробки веб-сайтів. Чітко сформульовані та верифіковані вимоги дозволяють створити якісний, надійний та зручний для користувачів веб-сайт, який відповідає потребам бізнесу та очікуванням користувачів.

1.1.3. Етапи процесу верифікації вимог

Процес верифікації вимог – це не одноразова дія, а скоріше циклічний процес, що складається з кількох послідовних етапів. Ці етапи забезпечують систематичний підхід до перевірки та підтвердження якості вимог, що, в свою чергу, сприяє успішній розробці веб-сайту. Хоча конкретні кроки можуть варіюватися залежно від методології розробки та специфіки проекту,

типовий процес верифікації включає наступні основні етапи: збір вимог, аналіз вимог, документування вимог, перевірка вимог та затвердження вимог.

Першим етапом є збір вимог. На цьому етапі відбувається активна взаємодія з замовником, користувачами та іншими зацікавленими сторонами з метою отримання повного та чіткого розуміння їхніх потреб та очікувань щодо майбутнього веб-сайту. Збір вимог може здійснюватися за допомогою різних методів, таких як інтерв'ю, опитування, воркшопи, аналіз документації та спостереження за користувачами. Важливо зафіксувати всі отримані вимоги, навіть якщо на перший погляд вони здаються несуттєвими.

Наступний етап – аналіз вимог. На цьому етапі зібрані вимоги ретельно аналізуються з метою виявлення можливих неточностей, суперечностей, неоднозначностей та неповноти. Аналіз вимог передбачає використання різних технік, таких як декомпозиція вимог, створення діаграм випадків використання, моделювання бізнес-процесів та аналіз ризиків. Метою аналізу є отримання чіткого та структурованого набору вимог, які є зрозумілими для всіх учасників проекту.

Після аналізу вимоги документуються. Документування вимог є важливим етапом, оскільки воно забезпечує збереження та доступність інформації про вимоги протягом усього життєвого циклу розробки. Документація вимог може мати різні форми, такі як текстові описи, таблиці, діаграми та прототипи. Важливо, щоб документація була чіткою, зрозумілою та легкою для читання. Існує багато стандартів та шаблонів для документування вимог, які можна використовувати для забезпечення консистентності та якості документації.

Після документування вимоги підлягають перевірці. Перевірка вимог – це процес оцінки якості документованих вимог на відповідність критеріям коректності, повноти, несуперечливості, однозначності, верифікованості та простежуваності. Перевірка може здійснюватися за допомогою різних

методів, таких як формальні та неформальні перегляди, інспекції, тестування прототипів та використання спеціалізованого програмного забезпечення. Метою перевірки є виявлення та усунення всіх помилок та недоліків у вимогах.

Останнім етапом є затвердження вимог. Після успішної перевірки вимоги затверджуються замовником та іншими зацікавленими сторонами. Затвердження вимог означає, що всі учасники проекту згодні з тим, що документовані вимоги є повними, коректними та відповідають потребам проекту. Затверджені вимоги стають основою для подальших етапів розробки веб-сайту. Важливо зазначити, що процес верифікації вимог може бути ітеративним, тобто деякі етапи можуть повторюватися кілька разів, особливо у випадку складних проектів або змін у вимогах. Зазвичай після етапу перевірки, якщо було знайдено помилки, відбувається повернення на етап аналізу або документування, для внесення коректив.

Таким чином, процес верифікації вимог є важливим інструментом для забезпечення якості та успіху розробки веб-сайтів. Він дозволяє виявити та усунути помилки на ранніх етапах, що значно зменшує витрати та ризики проекту.

1.2. Методи верифікації вимог

Верифікація вимог, як ми вже з'ясували, є критично важливим етапом у процесі розробки веб-сайтів. Для забезпечення ефективної верифікації існує широкий спектр методів та технік, які можна використовувати окремо або в комбінації, залежно від специфіки проекту, типу вимог та доступних ресурсів. Вибір конкретних методів верифікації залежить від багатьох факторів, включаючи складність проекту, доступний час, бюджет, кваліфікацію команди та тип вимог, що перевіряються.

Загалом, методи верифікації вимог можна поділити на дві основні категорії: ручні та автоматизовані. Ручні методи передбачають безпосередню

участь людей у процесі перевірки вимог, таких як аналітики, розробники, тестувальники та замовники. Ці методи часто базуються на інспекціях, обговореннях та аналізі документації. Автоматизовані методи, навпаки, використовують спеціалізоване програмне забезпечення для автоматизації деяких аспектів процесу верифікації, наприклад, для аналізу специфікацій на наявність суперечностей або для генерації тестових випадків.

Ручні методи верифікації є особливо корисними на ранніх етапах розробки, коли вимоги ще не є повністю формалізованими та потребують детального обговорення та уточнення з зацікавленими сторонами. Вони дозволяють виявити неточності, неоднозначності та суперечності у вимогах шляхом безпосередньої взаємодії між людьми. До таких методів належать, наприклад, перегляди вимог, технічні інспекції, інтерв'ю з замовниками та користувачами, створення прототипів та проведення воркшопів.

Автоматизовані методи верифікації, зі свого боку, є більш ефективними для перевірки великих обсягів формалізованих вимог та для виявлення формальних помилок, таких як синтаксичні помилки або порушення правил формальної специфікації. Вони дозволяють значно скоротити час, необхідний для перевірки вимог, та підвищити точність виявлення помилок. До таких методів належать, наприклад, використання інструментів для аналізу специфікацій, формальні методи верифікації, такі як моделювання та перевірка моделей, а також генерація тестових випадків на основі специфікацій.

Важливо зазначити, що ручні та автоматизовані методи верифікації не є взаємовиключними. На практиці часто використовується комбінований підхід, який поєднує переваги обох типів методів. Наприклад, ручні перегляди можуть бути використані для виявлення загальних проблем у вимогах, а автоматизовані методи – для перевірки формальних аспектів та генерації тестових випадків.

Ефективність верифікації вимог значною мірою залежить від правильного вибору методів та технік, а також від їхнього грамотного застосування. При виборі методів необхідно враховувати специфіку проекту, тип вимог, доступні ресурси та інші фактори. Важливо також забезпечити належну підготовку команди та наявність необхідних інструментів та засобів.

У наступних підпунктах ми детальніше розглянемо окремі методи верифікації вимог, їхні переваги та недоліки, а також приклади їхнього застосування у контексті розробки веб-сайтів. Це дозволить отримати повне уявлення про існуючі методи верифікації та зробити обґрунтований вибір при розробці конкретного проекту.

1.2.1. Ручні методи верифікації

Ручні методи верифікації відіграють важливу роль у процесі забезпечення якості вимог, оскільки вони дозволяють виявити широкий спектр проблем, які важко або неможливо виявити за допомогою автоматизованих методів. Ці методи базуються на безпосередній взаємодії між людьми, аналізі документації та обговоренні вимог.

Перегляд (рев'ю) – це систематичний процес оцінки документації вимог з метою виявлення помилок, неточностей, неоднозначностей та інших проблем. Перегляди можуть проводитися на різних етапах розробки та з різною метою. Існує кілька видів перегляду, які відрізняються за формальністю та інтенсивністю:

а) неформальний перегляд – це найпростіший вид перегляду, який зазвичай проводиться одним або двома колегами. Він передбачає швидкий огляд документації з метою виявлення очевидних помилок та неточностей. Неформальний перегляд є корисним для швидкого отримання зворотного зв'язку та виявлення грубих помилок на ранніх етапах. Перевагою є швидкість та простота організації, недоліком – низька гарантія виявлення всіх помилок;

б) технічний перегляд — цей вид перегляду є більш формальним та передбачає участь групи експертів, які детально аналізують документацію з технічної точки зору. Технічний перегляд фокусується на виявленні технічних помилок, таких як несумісність з іншими системами, порушення стандартів кодування або неефективні алгоритми. Перевагою є глибокий технічний аналіз, недоліком – потреба у кваліфікованих експертах та більші витрати часу;

в) інспекція — це найбільш формальний та структурований вид перегляду, який передбачає використання чітко визначеної процедури та контрольних списків. Інспекція проводиться групою спеціально навчених інспекторів та фокусується на виявленні всіх видів помилок, включаючи помилки у вимогах, дизайні, коді та тестових випадках. Перевагою є висока ефективність виявлення помилок, недоліком – значні витрати часу та ресурсів на підготовку та проведення.

1.2.2. Автоматизовані методи верифікації

Автоматизовані методи верифікації використовують програмне забезпечення для автоматизації процесів аналізу та перевірки вимог. Це дозволяє підвищити ефективність та точність верифікації, особливо при роботі з великими обсягами вимог. Ці методи особливо корисні для виявлення формальних помилок, таких як синтаксичні помилки, порушення правил формальної специфікації, суперечності та неповнота. Розглянемо детальніше деякі з найбільш поширених автоматизованих методів: аналіз специфікацій за допомогою спеціалізованого програмного забезпечення та використання формальних методів верифікації.

Аналіз специфікацій можливий за допомогою спеціалізованого програмного забезпечення. Існує ряд інструментів, які допомагають автоматично виявляти помилки та суперечності у вимогах, представлених у формалізованому вигляді. Ці інструменти можуть аналізувати специфікації,

написані на різних мовах, таких як UML, SysML, SDL, та інші. Вони виконують різноманітні перевірки, включаючи:

а) перевірка синтаксису та семантики — інструменти перевіряють, чи відповідають вимоги правилам синтаксису та семантики вибраної мови специфікацій. Це допомагає виявити помилки у формулюванні вимог та забезпечити їхню однозначність;

б) виявлення суперечностей — інструменти аналізують вимоги з метою виявлення логічних суперечностей між ними. Наприклад, якщо одна вимога стверджує, що користувач повинен мати можливість редагувати профіль, а інша – що редагування профілю заборонено, інструмент виявить цю суперечність;

в) перевірка повноти — інструменти можуть допомогти виявити неповноту у вимогах, наприклад, відсутність опису певних функціональностей або обробки помилкових ситуацій;

г) генерація тестових випадків — деякі інструменти здатні автоматично генерувати тестові випадки на основі специфікацій вимог. Це дозволяє значно скоротити час, необхідний для розробки тестів, та підвищити їхню якість.

Окрім спеціального програмного забезпечення, існують також формальні методи верифікації. Формальні методи верифікації базуються на використанні математичних методів та логіки для доведення коректності вимог. Ці методи дозволяють отримати строгі докази того, що система відповідає заданим специфікаціям. Хоча застосування формальних методів може бути досить складним та потребувати спеціальних знань, вони є особливо корисними для критично важливих систем, де помилки можуть мати серйозні наслідки. Формальні методи верифікації включають:

а) моделювання — створення формальної моделі системи на основі вимог. Ця модель може бути представлена у вигляді математичних формул, автоматів, діаграм станів або інших формальних нотацій;

б) перевірка моделей (Model checking) — автоматизована техніка, яка перевіряє, чи задовольняє модель системи заданим властивостям, вираженим у формальній логіці;

в) теоретичне доведення (Theorem proving) — використання математичних методів та логічних висновків для доведення того, що система відповідає заданим специфікаціям.

Застосування формальних методів верифікації вимагає високої кваліфікації та значних витрат часу, тому вони зазвичай використовуються для критично важливих систем, де вартість помилок є дуже високою. У контексті розробки веб-сайтів, повне формальне доведення коректності вимог зустрічається рідко, але використання окремих елементів формальних методів, таких як моделювання та перевірка моделей, може бути корисним для аналізу складних функціональностей та забезпечення їхньої коректності.

Автоматизовані методи верифікації, особливо в комбінації з ручними методами, дозволяють значно підвищити якість та надійність веб-сайтів, забезпечуючи відповідність вимогам та зменшуючи ризик виникнення помилок на пізніх етапах розробки.

1.2.3. Комбіновані методи верифікації

Успішна верифікація вимог часто потребує комплексного підходу, який поєднує переваги як ручних, так і автоматизованих методів. Комбіновані методи верифікації дозволяють досягти більш повного та якісного аналізу вимог, мінімізуючи недоліки окремих методів. Застосування комбінованого підходу дозволяє не тільки виявити ширший спектр помилок, але й оптимізувати процес верифікації з точки зору витрат часу та ресурсів.

Одним з поширених комбінованих підходів є поєднання ручних переглядів з автоматизованим аналізом специфікацій. Наприклад, на початковому етапі збору та документування вимог, коли вимоги ще не є повністю формалізованими, проводяться ручні перегляди з замовниками та користувачами. Ці перегляди дозволяють виявити загальні неточності, неоднозначності та суперечності у вимогах, а також отримати зворотний зв'язок від зацікавлених сторін. Після того, як вимоги формалізовані та представлені у вигляді специфікацій, наприклад, у UML або SysML, застосовуються автоматизовані інструменти для аналізу цих специфікацій. Автоматизований аналіз дозволяє виявити формальні помилки, такі як синтаксичні помилки, порушення правил формальної специфікації, логічні суперечності та неповноту. Таким чином, ручні перегляди забезпечують якісний аналіз змісту вимог, а автоматизований аналіз – формальну коректність специфікацій.

Іншим прикладом комбінованого підходу є поєднання створення прототипів з автоматизованою генерацією тестових випадків. На етапі створення прототипів розробники створюють інтерактивні моделі веб-сайту, які демонструють його основну функціональність та інтерфейс користувача. Ці прототипи використовуються для отримання зворотного зв'язку від користувачів та уточнення вимог. Після того, як вимоги уточнені та зафіксовані, застосовуються автоматизовані інструменти для генерації тестових випадків на основі цих вимог. Це дозволяє забезпечити повне покриття вимог тестами та підвищити якість тестування. Таким чином, прототипи забезпечують візуалізацію та розуміння вимог з боку користувачів, а автоматизована генерація тестів – ефективну перевірку реалізації цих вимог.

Ще один варіант комбінованого підходу передбачає використання формальних методів верифікації для аналізу критично важливих частин системи в поєднанні з ручними методами для аналізу менш критичних

частин. Наприклад, якщо веб-сайт містить складну систему авторизації, для перевірки її коректності можуть бути використані формальні методи, такі як перевірка моделей. Для перевірки інших, менш критичних функціональностей, можуть бути застосовані ручні перегляди та тестування. Це дозволяє оптимізувати витрати на верифікацію, зосередивши зусилля на найбільш важливих аспектах системи.

Важливо підкреслити, що вибір конкретних методів верифікації та їх комбінацій залежить від багатьох факторів, таких як складність проекту, тип вимог, доступний час, бюджет, кваліфікація команди та рівень ризику. Ефективне застосування комбінованих методів верифікації потребує ретельного планування та координації між різними учасниками проекту. Проте, комбінований підхід, як правило, забезпечує найкращі результати, дозволяючи досягти високої якості вимог та мінімізувати ризики, пов'язані з помилками у специфікаціях.

1.3. Існуючі інструменти та засоби для верифікації вимог

Для ефективної верифікації вимог існує широкий спектр інструментів та засобів, які допомагають на різних етапах процесу. Ці інструменти можна поділити на кілька категорій залежно від їхнього призначення: інструменти для управління вимогами, інструменти для моделювання та аналізу вимог, а також інструменти для автоматизованого тестування вимог.

1.3.1. Інструменти для управління вимогами

Інструменти для управління вимогами призначені для зберігання, відстеження та керування вимогами протягом усього життєвого циклу розробки. Вони забезпечують централізоване сховище для всіх вимог, дозволяють відстежувати зміни, встановлювати зв'язки між вимогами, а також керувати версіями та статусами вимог. Використання таких інструментів значно спрощує процес верифікації та забезпечує прозорість та контрольованість. Приклади таких інструментів:

а) Jira — популярний інструмент для управління проектами та відстеження помилок, який також має функціональність для управління вимогами. Jira дозволяє створювати завдання, пов'язані з вимогами, відстежувати їхній статус та прогрес виконання. Хоча Jira в основному інструмент для відстеження задач, за допомогою плагінів можливо розширити його функціонал для повноцінного управління вимогами;

б) Confluence — інструмент для спільної роботи та створення документації, який може бути використаний для зберігання та організації вимог. Confluence дозволяє створювати сторінки з описом вимог, додавати коментарі та відстежувати зміни. Інтеграція з Jira дозволяє пов'язувати вимоги з завданнями розробки;

в) IBM Rational DOORS — потужний інструмент для управління вимогами, який широко використовується в великих та складних проектах, особливо в галузях з високими вимогами до безпеки та надійності, таких як аерокосмічна, автомобільна та медична промисловість. DOORS забезпечує повний контроль над вимогами, включаючи відстеження змін, керування версіями, аналіз впливу змін та генерацію звітів;

г) Jama Connect — платформа для управління життєвим циклом розробки продукту, яка включає можливості для управління вимогами, тестування та управління ризиками. Jama Connect забезпечує простежуваність між вимогами, тестами та іншими артефактами розробки;

д) Visure Requirements — ще один потужний інструмент для управління вимогами, який підтримує різні методології розробки та забезпечує повний контроль над вимогами протягом усього життєвого циклу.

1.3.2. Інструменти для моделювання та аналізу вимог

Інструменти для моделювання та аналізу вимог використовуються для створення діаграм, моделей та інших візуальних представлень вимог.

Візуалізація вимог допомагає краще зрозуміти їхню структуру, взаємозв'язки та поведінку системи. Приклади таких інструментів:

а) Enterprise Architect — потужний інструмент для моделювання бізнес-процесів, програмного забезпечення та системної архітектури, який підтримує різні мови моделювання, такі як UML, SysML та BPMN. Enterprise Architect дозволяє створювати різні типи діаграм, такі як діаграми класів, діаграми випадків використання, діаграми послідовностей та діаграми станів;

б) Visual Paradigm — ще один популярний інструмент для моделювання, який підтримує UML, SysML та інші мови моделювання. Visual Paradigm пропонує широкий спектр функцій для створення діаграм, генерації коду та документації;

в) Draw.io — безкоштовний онлайн-інструмент для створення діаграм різних типів, включаючи UML-діаграми. Draw.io є простим у використанні та не потребує встановлення.

Використання цих інструментів дозволяє візуалізувати вимоги, що сприяє кращому розумінню та комунікації між учасниками проекту.

1.4. Проблеми та обмеження сучасних методів верифікації вимог

Незважаючи на існування різноманітних методів та інструментів верифікації вимог, цей процес залишається складним та пов'язаним з певними проблемами та обмеженнями. Ці проблеми можуть впливати на ефективність верифікації та призводити до помилок у вимогах, що, в свою чергу, може негативно позначитися на якості кінцевого продукту. Розглянемо детальніше деякі з найбільш значущих проблем та обмежень: труднощі збору повних та точних вимог, складність автоматизації процесу верифікації та людський фактор у процесі верифікації.

1.4.1. Труднощі збору повних та точних вимог

Однією з ключових проблем верифікації є труднощі збору повних та точних вимог на початкових етапах проекту. Ця проблема часто пов'язана з комунікаційними бар'єрами між замовниками та розробниками. Замовники можуть не завжди чітко усвідомлювати свої потреби або не вміти їх правильно сформулювати. Розробники, зі свого боку, можуть неправильно інтерпретувати вимоги замовників або не врахувати всі нюанси предметної області. Крім того, вимоги можуть змінюватися протягом проекту під впливом різних факторів, таких як зміни в бізнес-середовищі, поява нових технологій або зворотний зв'язок від користувачів. Ці зміни можуть призвести до невідповідностей між вимогами та реалізованою системою, якщо вони не будуть своєчасно відстежені та враховані. Зміни в вимогах, якщо їх не контролювати належним чином, можуть призвести до значних витрат часу та ресурсів на переробку вже реалізованих функціональностей. Тому, ефективне управління змінами вимог є критично важливим для успішної верифікації.

1.4.2. Складність автоматизації процесу верифікації

Хоча автоматизовані методи верифікації мають багато переваг, повна автоматизація цього процесу залишається складною задачею. Існуючі інструменти та методи автоматизації часто мають обмеження щодо типу вимог, які вони можуть обробляти. Наприклад, деякі інструменти можуть ефективно аналізувати формалізовані вимоги, представлені у вигляді специфікацій, але вони можуть бути менш ефективними для аналізу неформальних вимог, описаних природною мовою. Крім того, автоматизовані методи часто потребують значних зусиль на підготовку даних та налаштування інструментів. Розробка формальних специфікацій, придатних для автоматизованого аналізу, також може бути трудомістким процесом. Тому, на практиці часто використовується комбінований підхід, який поєднує автоматизовані методи з ручними перевірками.

1.4.3. Людський фактор у процесі верифікації

Людський фактор відіграє важливу роль у процесі верифікації вимог. Якість верифікації значною мірою залежить від кваліфікації та досвіду фахівців, що займаються цим процесом. Недостатня кваліфікація або неуважність фахівців можуть призвести до пропуску помилок у вимогах. Крім того, суб'єктивність людського фактору може впливати на інтерпретацію вимог та результати перевірок. Тому, для забезпечення ефективної верифікації необхідно залучати кваліфікованих та досвідчених фахівців, які мають глибоке розуміння предметної області та володіють необхідними методами та інструментами верифікації. Важливим є також забезпечення належної комунікації та співпраці між усіма учасниками процесу верифікації.

1.5. Постановка задачі

Виходячи з аналізу існуючих методів і інструментів для верифікації вимог до веб-сайтів, представлених у попередніх підрозділах, було визначено основні проблеми, які потребують вирішення. Наразі є потреба у дослідженні, яке буде зосереджено на проблемах верифікації вимог на початковому етапі. Дослідження проводиться з метою подальшої розробки програмного забезпечення для автоматизації верифікації вимог до веб-сайтів.

Дослідження буде включати в себе:

а) вибір методів верифікації вимог до сайтів;

б) вибір показників ефективності роботи методів;

в) створення процедури проведення експериментів;

г) проведення експериментів та вимірювання показників;

д) порівняння показників.

Висновки до розділу 1

У цьому розділі було проведено аналіз сучасних методів та інструментів верифікації вимог до веб-сайтів. Розглянуто різні підходи, включаючи ручні, автоматизовані та комбіновані методи, а також існуючі програмні засоби для управління, моделювання та тестування вимог. Аналіз показав, що кожен з підходів має свої сильні та слабкі сторони.

Ручні методи, такі як перегляди, інтерв'ю та створення прототипів, дозволяють отримати глибоке розуміння потреб замовника та виявити широкий спектр проблем, пов'язаних з нечіткими або суперечливими формулюваннями вимог. Однак, вони є трудомісткими та залежними від людського фактору. Автоматизовані методи, зі свого боку, забезпечують швидкий та точний аналіз формалізованих вимог, але мають обмеження щодо обробки неструктурованої інформації та потребують значних зусиль на підготовку даних. Комбіновані методи дозволяють поєднати переваги обох підходів, але потребують ретельного планування та координації.

Аналіз існуючих інструментів показав наявність широкого спектру програмних засобів для підтримки процесу верифікації вимог. Інструменти для управління вимогами забезпечують централізоване зберігання та відстеження вимог протягом усього життєвого циклу розробки. Інструменти для моделювання та аналізу вимог дозволяють візуалізувати вимоги та краще зрозуміти їхню структуру та взаємозв'язки. Інструменти для автоматизованого тестування дозволяють перевірити відповідність реалізованого веб-сайту заданим вимогам.

Незважаючи на значний прогрес у галузі верифікації вимог, існують певні проблеми та обмеження, які потребують подальшого дослідження. Однією з ключових проблем є труднощі збору повних та точних вимог на початковому етапі проекту, коли відбувається налагодження зв'язку з клієнтом. Нечіткі або суперечливі формулювання вимог, непорозуміння між замовниками та

розробниками, а також зміни в вимогах протягом проекту можуть призвести до серйозних проблем на пізніх етапах розробки.

Саме тому, подальше дослідження буде зосереджено на проблемах верифікації вимог на початковому етапі, коли відбувається комунікація з клієнтом. Особлива увага буде приділена розробці інструментів для автоматизованого аналізу тексту вимог до веб-сайтів.

РОЗДІЛ 2. МЕТОДИКА ТА ДОСЛІДЖЕННЯ

Розробка якісного та успішного веб-сайту вимагає ретельного підходу до всіх етапів життєвого циклу програмного забезпечення, починаючи від збору вимог і закінчуючи тестуванням та введенням в експлуатацію. Одним з найважливіших етапів, що безпосередньо впливає на кінцевий результат, є верифікація вимог. Верифікація, на відміну від валідації, яка перевіряє, чи створюється правильний продукт, фокусується на тому, чи правильно створюється продукт, тобто чи відповідає розроблений продукт заданим вимогам. Нечітко сформульовані, суперечливі або неповні вимоги можуть призвести до серйозних проблем на подальших етапах розробки, таких як непорозуміння між замовником та розробниками, збільшення термінів розробки, перевитрати бюджету та, зрештою, до створення продукту, який не задовольняє потреби користувачів.

Саме тому ефективна верифікація вимог є критично важливою для забезпечення якості програмного забезпечення, зокрема веб-сайтів. Цей розділ присвячено дослідженню методів верифікації вимог до веб-сайтів та розробці відповідної методики. У ньому буде розглянуто існуючі підходи до верифікації, їхні переваги та недоліки, а також запропоновано удосконалений підхід, що враховує специфіку веб-розробки.

Верифікація вимог – це процес перевірки того, чи є вимоги повними, несуперечливими, однозначними, зрозумілими, досяжними та такими, що піддаються тестуванню. Цей процес включає в себе різноманітні техніки та методи, спрямовані на виявлення потенційних проблем у специфікаціях вимог. Ефективна верифікація дозволяє запобігти помилкам на ранніх стадіях розробки, що значно зменшує витрати на їхнє виправлення в майбутньому.

У цьому розділі ми розглянемо різні методи верифікації, починаючи від традиційних ручних методів, таких як інспектування та перевірки, і закінчуючи автоматизованими підходами. Ручні методи, хоча і є важливими

для виявлення певних типів помилок, можуть бути трудомісткими та суб'єктивними. Автоматизовані методи, з іншого боку, дозволяють обробляти великі обсяги інформації та виявляти помилки більш ефективно та об'єктивно.

Особливу увагу в цьому розділі буде приділено аналізу семантичних проблем у вимогах, таких як нечіткі формулювання, неоднозначності та суб'єктивні оцінки. Ці проблеми часто залишаються непоміченими при використанні традиційних методів верифікації, але можуть призвести до серйозних непорозумінь та помилок на етапах проектування та розробки.

2.1. Огляд існуючих методів верифікації вимог

Верифікація вимог є критично важливим етапом у процесі розробки програмного забезпечення, включаючи веб-сайти. Вона забезпечує відповідність вимог потребам замовника та користувачів, а також їхню коректність, повноту, несуперечливість та однозначність. Існує широкий спектр методів верифікації, які можна класифікувати як ручні та автоматизовані. Кожен з цих підходів має свої переваги та недоліки, а також області застосування. У цьому підрозділі ми детально розглянемо ці методи, зосереджуючись на ручному аналізі, як одному з найпоширеніших, та автоматизованих підходах з використанням LanguageTool та NLP (зокрема, бібліотеки spaCy для Python).

2.1.1. Ручний аналіз вимог

Ручний аналіз вимог є традиційним, але досі широко використовуваним методом верифікації, що базується на безпосередній перевірці тексту вимог людьми – експертами, аналітиками, замовниками або іншими зацікавленими сторонами. Цей процес передбачає уважне прочитання та осмислення кожної вимоги з метою виявлення потенційних проблем та невідповідностей. Ручний аналіз покладається на людський інтелект, досвід та знання предметної

області, що дозволяє виявляти не лише формальні помилки, але й семантичні неточності, неоднозначності та суперечності.

Існує кілька технік, що застосовуються в рамках ручного аналізу. Інспектування є формальним процесом, де група експертів ретельно переглядає документ з вимогами, дотримуючись чітко визначених процедур та критеріїв. Кожен учасник інспектування виконує певну роль, наприклад, модератора, автора, інспектора або секретаря, що забезпечує структурований та організований підхід до перевірки. Огляди, на відміну від інспектування, є менш формальними та передбачають обмін відгуками та коментарями між зацікавленими сторонами. Огляди можуть проводитися у різних форматах, від неформальних зустрічей до письмових рецензій. Проходження сценаріїв передбачає моделювання використання системи на основі сформульованих вимог. Учасники цього процесу крок за кроком проходять через різні сценарії, аналізуючи, як система повинна поводитися в кожній конкретній ситуації, що дозволяє виявити потенційні проблеми з функціональністю та інтерфейсом. Використання контрольних списків є ще однією корисною технікою ручного аналізу. Контрольні списки містять перелік питань або критеріїв, які використовуються для систематичної перевірки кожної вимоги на відповідність певним стандартам якості.

Переваги ручного аналізу:

а) Гнучкість — ручний аналіз дозволяє враховувати контекст та складні ситуації, які можуть бути важко формалізовані для автоматизованої перевірки;

б) виявлення неявних помилок — експерти можуть виявляти неявні помилки, неоднозначності та невідповідності, які неможливо знайти за допомогою простих алгоритмів;

в) залучення зацікавлених сторін — ручний аналіз сприяє залученню зацікавлених сторін до процесу верифікації, що покращує комунікацію та порозуміння.

Недоліки ручного аналізу:

а) суб'єктивність — результати ручного аналізу залежать від досвіду, знань та уважності експертів, що може призвести до суб'єктивних оцінок та пропуску помилок;

б) трудомісткість — ручний аналіз вимагає значних витрат часу та ресурсів, особливо для великих та складних проєктів;

в) схильність до помилок — людський фактор може призвести до помилок, неуважності та пропуску важливих деталей.

2.1.2. Автоматизовані методи верифікації

Автоматизовані методи верифікації вимог використовують програмне забезпечення та алгоритми для автоматичної перевірки тексту вимог на відповідність певним критеріям якості. Це дозволяє підвищити ефективність, об'єктивність та швидкість процесу верифікації, особливо при роботі з великими обсягами документації. У цьому дослідженні ми розглянемо два основні автоматизовані підходи: використання LanguageTool для перевірки орфографії та стилістики, а також застосування методів обробки природної мови (NLP) з використанням бібліотеки spaCy для Python для виявлення семантичних проблем.

2.1.2.1. Використання LanguageTool

LanguageTool – це потужний інструмент з відкритим кодом, призначений для перевірки граматики, стилю, пунктуації та орфографії тексту. Він підтримує велику кількість мов, включаючи українську, що робить його цінним ресурсом для верифікації вимог, написаних українською мовою. LanguageTool працює на основі набору правил та словників, що дозволяє

йому виявляти різноманітні помилки, від простих друкарських помилок до складніших граматичних конструкцій та стилістичних неточностей.

Переваги використання LanguageTool:

а) виявлення орфографічних та граматичних помилок — LanguageTool ефективно виявляє друкарські помилки, неправильне вживання слів, помилки в узгодженні часів та інші граматичні неточності, що сприяє підвищенню якості тексту вимог;

б) перевірка стилю та пунктуації — інструмент допомагає виявити стилістичні помилки, такі як надмірне використання складних речень, тавтології або неправильне вживання розділових знаків;

в) підтримка багатьох мов — LanguageTool підтримує велику кількість мов, що робить його універсальним інструментом для верифікації вимог, написаних різними мовами;

г) інтеграція з різними платформами — LanguageTool може бути використаний як окрема програма, плагін для браузера або інтегрований в інші текстові редактори та IDE;

д) відкритий код — будучи програмним забезпеченням з відкритим кодом, LanguageTool є безкоштовним та доступним для використання та модифікації.

Недоліки використання LanguageTool:

а) обмеженість семантичним аналізом — LanguageTool зосереджений на формальних аспектах мови та не здатний виявляти складні семантичні помилки, такі як неоднозначності, неточності або суперечності в змісті вимог;

б) можливість хибних спрацювань — як і будь-який автоматизований інструмент, LanguageTool може генерувати хибні спрацювання, тобто позначати коректні фрагменти тексту як помилкові;

в) потреба в налаштуванні — для досягнення оптимальних результатів може знадобитися налаштування правил та словників LanguageTool відповідно до специфіки проєкту та використовуваної термінології.

2.1.2.2. Використання NLP (spaCy Python)

Обробка природної мови (NLP) – це галузь штучного інтелекту, що займається аналізом та розумінням людської мови. Бібліотека spaCy для Python є потужним інструментом для NLP, що надає широкий спектр функцій для обробки тексту, включаючи токенізацію, лематизацію, визначення частин мови, розпізнавання іменованих сутностей та інші. У контексті верифікації вимог, NLP може бути використано для розробки спеціалізованих аналізаторів, здатних виявляти семантичні проблеми, такі як неточності, неоднозначності та суб'єктивні оцінки.

Переваги використання NLP (spaCy Python):

а) виявлення семантичних помилок — за допомогою NLP можна розробити аналізатори, що здатні виявляти неточності, неоднозначності, суперечності та інші семантичні проблеми, які важко виявити за допомогою простих формальних методів;

б) аналіз контексту — NLP дозволяє аналізувати контекст вживання слів та фраз, що сприяє більш точному виявленню помилок та зменшенню кількості хибних спрацювань;

в) гнучкість та розширюваність — spaCy є потужним та гнучким інструментом, що дозволяє розробляти кастомні рішення для верифікації вимог з урахуванням специфіки проєкту;

г) можливість автоматизації складних перевірок — за допомогою NLP можна автоматизувати складні перевірки, такі як перевірка на повноту, несуперечливість та відповідність стандартам.

Недоліки використання NLP (spaCy Python):

а) складність розробки — розробка ефективних NLP-аналізаторів вимагає значних знань в галузі NLP та програмування;

б) потреба у навчальних даних — для навчання моделей NLP часто потрібні великі обсяги розмічених даних, що може бути проблематично для деяких предметних областей;

в) обчислювальні витрати — аналіз тексту за допомогою NLP може бути обчислювально витратним, особливо для великих обсягів даних;

г) можливість помилок — хоча NLP дозволяє враховувати контекст, він все ще може генерувати помилки, особливо при аналізі складних та неоднозначних текстів.

2.2. Методика дослідження

Ефективна верифікація вимог є критично важливим етапом у процесі розробки будь-якого програмного забезпечення, зокрема й веб-сайтів. Вона забезпечує відповідність кінцевого продукту потребам замовника та користувачів, мінімізує ризики виникнення помилок на пізніх стадіях розробки та сприяє оптимізації витрат. Існує безліч методів верифікації, кожен з яких має свої переваги та недоліки. Для об'єктивної оцінки ефективності цих методів необхідно провести ретельне дослідження, що дозволить порівняти їхні можливості та визначити сфери застосування.

Цей підрозділ описує детальну методику проведеного дослідження, метою якого є порівняння ефективності різних підходів до верифікації вимог до веб-сайтів. Дослідження базується на проведенні експерименту, в якому використовувався спеціально підготовлений текст вимог, що містив навмисно внесені помилки різних типів. Такий підхід дозволяє створити контрольовані умови та об'єктивно оцінити здатність кожного методу виявляти ці помилки.

Метою цього дослідження є не лише порівняння ефективності різних методів верифікації, але й надання практичних рекомендацій щодо їхнього застосування в реальних проєктах. Результати дослідження допоможуть розробникам та аналітикам обирати найбільш підходящі методи верифікації в залежності від конкретних потреб та контексту проєкту, що сприятиме підвищенню якості програмного забезпечення та зменшенню ризиків.

2.2.1. Показники оцінки ефективності методів

Для об'єктивного порівняння ефективності різних методів верифікації вимог було визначено набір кількісних та якісних показників. Ці показники дозволяють оцінити здатність кожного методу виявляти різні типи помилок, а також витрачений на це час. Використання комплексу показників забезпечує всебічний аналіз та дозволяє зробити обґрунтовані висновки щодо переваг та недоліків кожного підходу.

Кількість знайдених орфографічних помилок, $N_{\text{орф}}$ — відображає здатність методу виявляти помилки в написанні слів, друкарські помилки, неправильне вживання розділових знаків та інші орфографічні неточності. Чим більше значення показнику, тим ефективнішим є метод у виявленні орфографічних помилок. Цей показник особливо важливий для оцінки ефективності LanguageTool, оскільки цей інструмент спеціалізується на виявленні саме таких помилок.

Кількість знайдених неточностей, $N_{\text{неточн}}$ — оцінює здатність методу виявляти нечіткі формулювання, відсутність конкретних деталей, використання загальних фраз, що можуть бути інтерпретовані по-різному. Формулювання є нечітким, якщо його не можливо точно виміряти (наприклад, замість конкретного строку виконання вимог з датами та часом, вказується «завтра». «Завтра» є неточним формулюванням). Виявлення неточностей вимагає глибшого розуміння контексту та предметної області,

тому цей показник важливий для оцінки ефективності ручного аналізу та NLP-аналізатора.

Кількість знайдених неоднозначностей, $N_{\text{неодн}}$ — відображає здатність методу виявляти формулювання, що мають кілька можливих інтерпретацій, що може призвести до непорозумінь між замовником та розробниками. Формулювання є неоднозначним, якщо воно має більше одного трактувань, таке як «тяжкий». Різниця між неточним формулюванням полягає у тому, неточне формулювання може мати лише одне трактування, і навпаки (наприклад, «завтра» означає проміжок часу наступного дня від 00:00 до 23:59, і це єдине значення, яке воно має. Натомість слово «тяжкий» може бути використано як міра ваги об’єкту, або як міра складності роботи). Як і у випадку з неточностями, виявлення неоднозначностей потребує аналізу контексту та розуміння можливих інтерпретацій, тому цей показник важливий для оцінки ручного аналізу та NLP-аналізатора.

Кількість знайдених суб'єктивних оцінок, $N_{\text{суб}}$ — оцінює здатність методу виявляти використання суб'єктивних термінів, таких як "зручний", "простий", "красивий", без чітких критеріїв оцінки. Виявлення суб'єктивних оцінок важливе для забезпечення чіткості та вимірюваності вимог, тому цей показник також важлива для оцінки ручного аналізу та NLP-аналізатора.

Кількість часу, витраченого на верифікацію вимог до сайту, $t_{\text{сер}}$ — вимірює час, витрачений на верифікацію тексту вимог кожним методом. Для ручного аналізу враховувався середній час, витрачений усіма учасниками експерименту. Цей показник дозволяє оцінити трудомісткість кожного методу та порівняти їхню ефективність з точки зору витрат часу. Важливо зазначити, що час, витрачений на розробку NLP-аналізатора, не враховувався в цій метриці, оскільки метою було порівняти час, необхідний для безпосередньої верифікації тексту.

Використання цих показників дозволить провести комплексний аналіз ефективності різних методів верифікації вимог та зробити обґрунтовані висновки щодо їхніх переваг та недоліків у різних аспектах.

2.2.2. Вхідні дані

Для проведення експерименту з порівняння ефективності методів верифікації вимог було підготовлено спеціальний набір вхідних даних – текст, що містить вимоги до умовного веб-сайту. Цей текст було розроблено з метою максимально наблизити його до реальних документів з вимогами, які використовуються в процесі розробки програмного забезпечення. Важливою особливістю підготовленого тексту є навмисне внесення помилок різних типів, що дозволило створити контрольовані умови для оцінки здатності різних методів верифікації виявляти ці помилки.

Текст вимог описував функціональність та характеристики простого веб-сайту, що дозволило зосередитися на аспектах верифікації, не ускладнюючи експеримент надмірною складністю предметної області. Вимоги були сформульовані українською мовою, що відповідає контексту дослідження.

Для забезпечення об'єктивності оцінки та можливості кількісного аналізу результатів, до тексту вимог до сайту було внесено на класифіковано помилки, відповідно до таблиці 2.1.

Таблиця 2.1 – Приклад класифікації та кількості помилок у тексті

	Орфографічна помилка, $N_{\text{орф}}$	Неточність, $N_{\text{неточн}}$	Неоднозначність, $N_{\text{неодн}}$	Суб'єктивна оцінка, $N_{\text{суб}}$
Кількість	11	5	3	7

2.2.3 Процедура проведення експерименту

Для досягнення мети дослідження – порівняння ефективності різних методів верифікації вимог – було розроблено та реалізовано чітку процедуру

експерименту. Ця процедура складалася з кількох послідовних етапів, кожен з яких відігравав важливу роль у забезпеченні об'єктивності та достовірності отриманих результатів.

На першому етапі було створено текст, що містив вимоги до умовного веб-сайту. Цей текст було розроблено таким чином, щоб він відображав типові вимоги, що зустрічаються в реальних проектах, охоплюючи різні аспекти функціональності, інтерфейсу, продуктивності та інші. Важливою умовою було створення тексту, який би був достатньо зрозумілим та конкретним для проведення верифікації, але водночас містив потенційні місця для виникнення помилок. Детальний опис тексту вимог наведено в пункті 2.2.2.

Після створення базового тексту вимог, до нього було навмисно внесено помилки різних типів. Це було зроблено для того, щоб створити контрольовані умови для оцінки здатності різних методів верифікації виявляти ці помилки. Помилки були класифіковані за такими категоріями: орфографічні помилки, неточності, неоднозначності, суб'єктивні оцінки та слова, що вимагають уточнення. Кількість помилок кожного типу була заздалегідь визначена та задокументована, що дозволило згодом порівняти результати верифікації з еталонним набором помилок. Детальний опис типів та кількості внесених помилок наведено в підпункті 2.2.2.

Підготовлений текст вимог з внесеними помилками було верифіковано трьома різними методами:

а) Ручний аналіз: для проведення ручного аналізу було сформовано групу з 8 учасників, які мали досвід у сфері аналізу вимог. Кожен учасник отримав текст вимог та інструкції щодо проведення верифікації. Учасники самостійно аналізували текст, фіксуючи всі знайдені помилки та витрачений на це час. Після завершення аналізу результати кожного учасника були зібрані та усереднені;

б) LanguageTool: для автоматизованої перевірки орфографії та стилістики було використано програму LanguageTool. Текст вимог було завантажено в програму, та отримані результати (перелік знайдених помилок) було задокументовано;

в) NLP-аналізатор (spaCy Python): для виявлення неточностей, неоднозначностей та суб'єктивних оцінок було розроблено спеціалізований NLP-аналізатор на основі бібліотеки spaCy для Python. Текст вимог було проаналізовано за допомогою цього аналізатора, та отримані результати (перелік знайдених проблем) було задокументовано.

Збір та аналіз даних: На заключному етапі було зібрано дані, отримані в результаті верифікації вимог різними методами. Ці дані включали:

а) кількість знайдених помилок кожного типу (орфографічні помилки, неточності, неоднозначності, суб'єктивні оцінки) для кожного методу;

б) час, витрачений на верифікацію вимог кожним методом (для ручного аналізу враховувався середній час, витрачений усіма учасниками).

Зібрані дані було проаналізовано з використанням кількісних методів, що дозволило порівняти ефективність різних методів верифікації та зробити обґрунтовані висновки.

2.3. Результати дослідження

2.3.1. Текст вимог до сайту

Текст вимог до сайту, який було використано для дослідження, зазначено у Додатку Г.

2.3.2. Результат верифікації ручним методом

Для проведення ручної верифікації було залучено 8 учасників, які мали досвід у сфері аналізу вимог. Кожен учасник самостійно аналізував текст вимог, фіксуючи всі знайдені помилки та витрачений на це час. Після

завершення аналізу результати кожного учасника було зібрано та усереднено, відповідно до таблиці 2.2.

Таблиця 2.2 — Результат верифікації ручним методом

Учасник	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
1	9	3	2	5	25
2	10	4	2	6	30
3	8	2	1	4	20
4	11	3	3	5	28
5	9	4	2	6	27
6	10	3	2	5	32
7	8	2	1	3	18
8	11	4	3	6	29
Середнє значення	9,5	3,125	2	5	26,125

2.3.3. Результат верифікації LanguageTool методом

Для верифікації тексту вимог за допомогою LanguageTool, текст було завантажено в програму, та отримані результати (перелік знайдених помилок) було задокументовано, відповідно до таблиці 2.3.

Таблиця 2.3 — Результат верифікації LanguageTool методом

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Кількість	11	0	0	0	1

2.3.4. Результат верифікації NLP-аналізатор (spaCy Python)

Для верифікації тексту вимог за допомогою NLP-аналізатора на основі spaCy Python, текст було проаналізовано за допомогою цього аналізатора, та отримані результати (перелік знайдених проблем) було задокументовано, відповідно до таблиці 2.4.

Таблиця 2.4 — Результат верифікації NLP-аналізатор (spaCy Python)

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Кількість	0	3	2	5	1

2.3.5. Загальні результати верифікації

Було зроблено підсумок всіх верифікацій та структуровано до таблиці 2.5.

Таблиця 2.5 – Результат верифікації всіх методів

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Ручний метод	9,5	3,125	2	5	26,125
LanguageTool	11	0	0	0	1
NLP-аналізатор (spaCy Python)	0	3	2	5	1

Порівняння методів за орфографічною помилкою представлено у вигляді діаграми, рисунок 2.1:

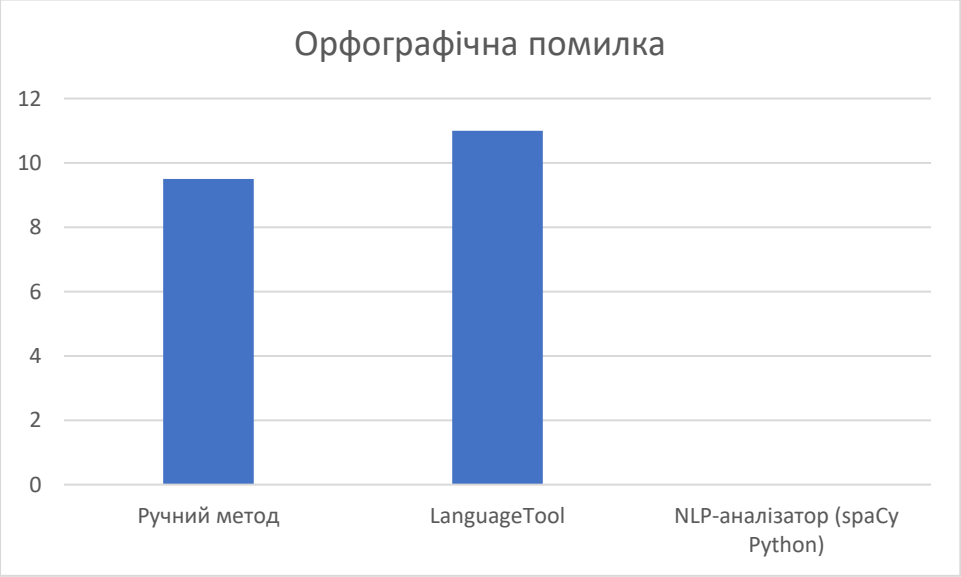

Рисунок 2.1 — Порівняння методів за орфографічною помилкою

Порівняння методів за помилкою неточності представлено у вигляді діаграми, рисунок 2.2:

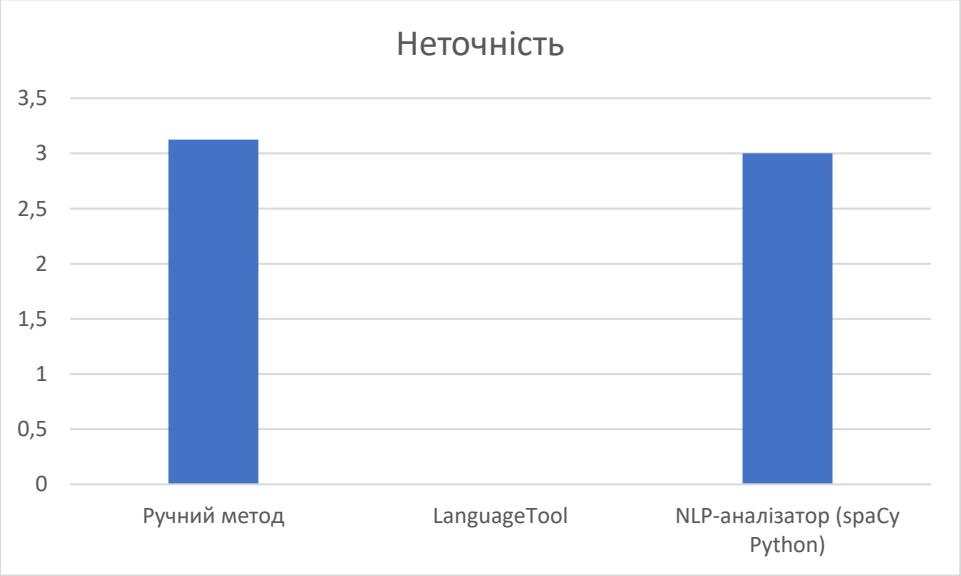

Рисунок 2.2 — Порівняння методів за помилкою неточності

Порівняння методів за помилкою неоднозначності представлено у вигляді діаграми, рисунок 2.3:

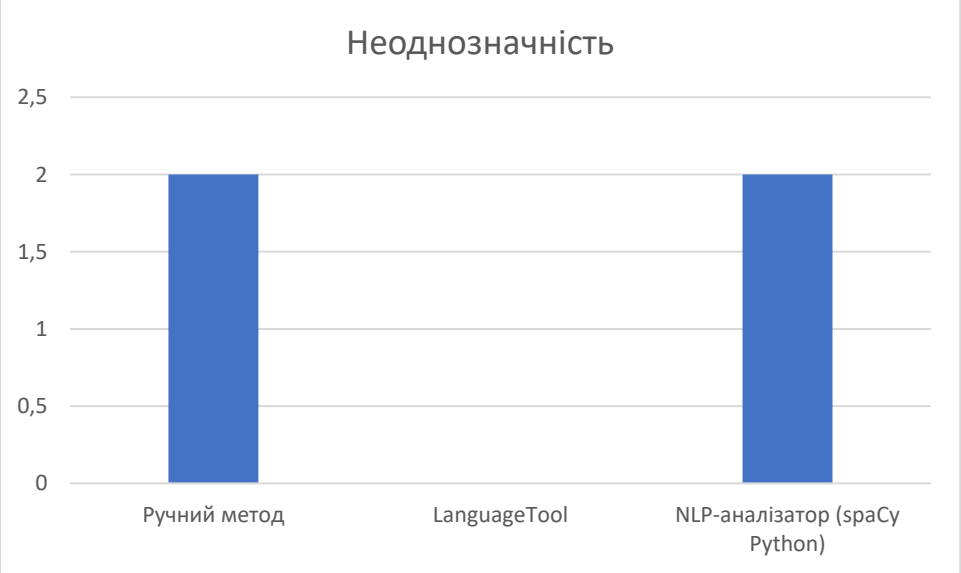

Рисунок 2.3 — Порівняння методів за помилкою неоднозначності

Порівняння методів за суб’єктивною оцінкою представлено у вигляді діаграми, рисунок 2.4:

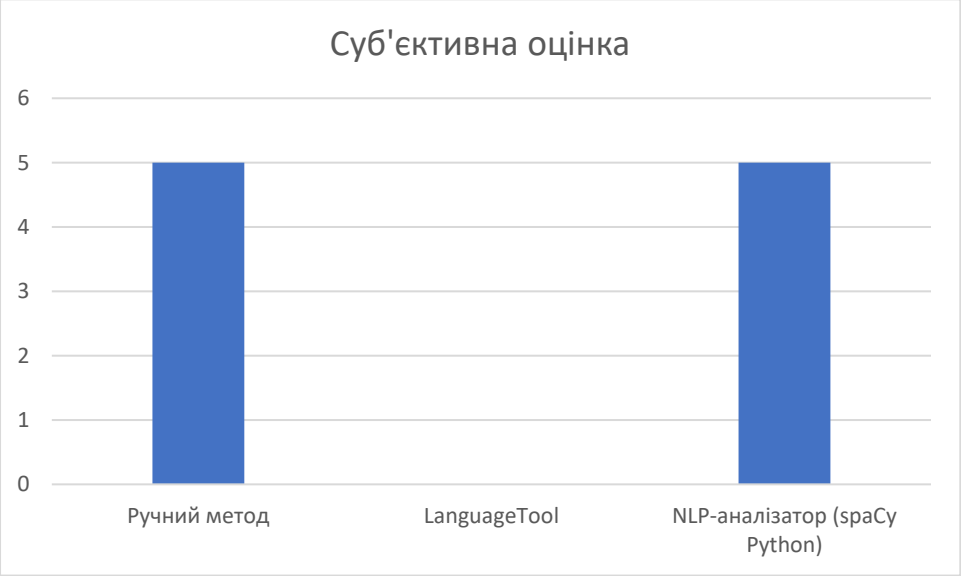

Рисунок 2.4 — Порівняння методів за суб’єктивною оцінкою

Порівняння методів за часом роботи представлено у вигляді діаграми, рисунок 2.5:

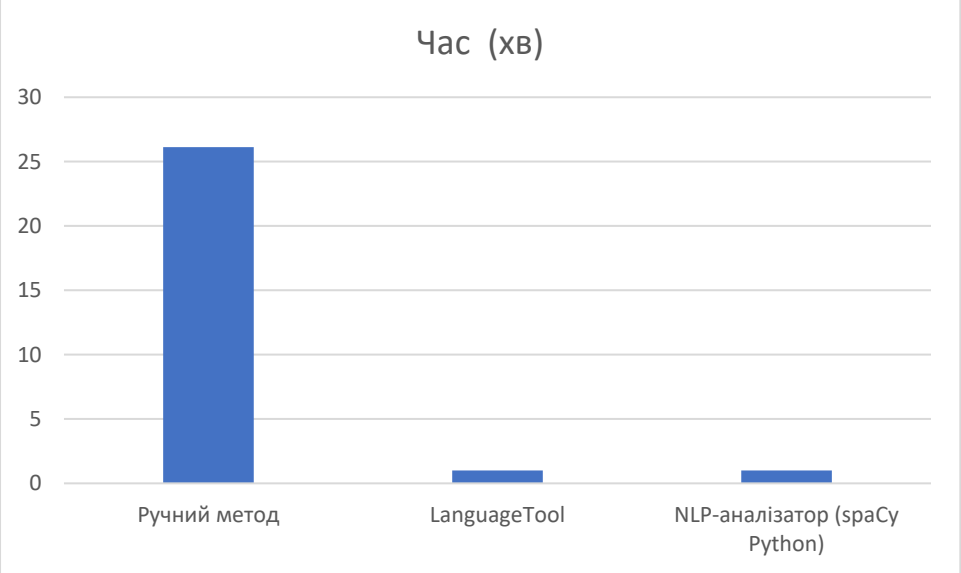

Рисунок 2.5 — Порівняння методів за часом роботи

Порівняння результатів верифікації різними методами дозволяє зробити наступні висновки:

а) орфографічні помилки — LanguageTool виявився найбільш ефективним методом для виявлення орфографічних помилок, хоча ручний метод також показав досить хороший результат;

б) неточності, неоднозначності та суб'єктивні оцінки — NLP-аналізатор виявився найбільш ефективним методом для виявлення цих типів помилок. Ручний метод також зміг виявити частину цих помилок, але з меншою ефективністю;

в) час верифікації — LanguageTool показав найшвидший час верифікації, тоді як ручний метод був найбільш трудомістким. NLP-аналізатор займає проміжне положення через час, що необхідний на написання програми.

Загалом, результати експерименту підтверджують, що кожен метод має свої сильні та слабкі сторони. LanguageTool є ефективним для швидкої перевірки орфографії, ручний метод дозволяє виявити широкий спектр помилок, але є трудомістким, а NLP-аналізатор є корисним для виявлення складніших семантичних проблем і може бути ефективним з точку зори співвідношення отриманого результату та часу. Комбінування методів, наприклад використання LanguageTool для попередньої перевірки орфографії та подальшого аналізу за допомогою NLP-аналізатора або ручного методу, може забезпечити найбільш повну та ефективну верифікацію вимог.

Висновки до розділу 2

У цьому розділі було розглянуто методи верифікації вимог до веб-сайтів, зокрема ручний аналіз, використання LanguageTool та розроблений NLP-аналізатор на основі бібліотеки spaCy. Проведене дослідження, що включало симуляцію ручного аналізу з учасниками та застосування автоматизованих інструментів, дозволило порівняти ефективність кожного методу у виявленні різних типів помилок: орфографічних, неточностей, неоднозначностей та суб'єктивних оцінок. Результати показали, що LanguageTool є найбільш ефективним для виявлення орфографічних помилок, тоді як NLP-аналізатор

краще справляється з виявленням семантичних проблем, таких як неточності, неоднозначності та суб'єктивні оцінки. Ручний аналіз, хоча й дозволяє виявити широкий спектр помилок, є більш трудомістким та суб'єктивним. Отже, дослідження підкреслило доцільність використання комбінованого підходу до верифікації вимог, що поєднує переваги різних методів для досягнення найбільш повного та ефективного результату.

РОЗДІЛ 3. ПРОЕКТУВАННЯ Й РОЗРОБКА

3.1. Зовнішнє проектування

3.1.1. Функціональне призначення

Програма призначена для автоматичної верифікації вимог до веб-сайтів у вигляді тексту. Основні функції включають:

а) виявлення орфографічних і граматичних помилок;

б) пошук неточних формулювань, неоднозначностей, суб’єктивних оцінок;

в) генерація звітів із виявленими помилками;

г) візуалізація аналізу за допомогою інтерактивного графічного інтерфейсу користувача (GUI).

3.1.2. Експлуатаційне призначення

Система створена для використання аналітиками, тестувальниками та розробниками, які працюють із вимогами до сайтів. Програма забезпечує автоматизацію процесу перевірки вимог, підвищуючи ефективність і якість аналізу за рахунок зекономленого часу.

3.1.3. Функціональні вимоги

Програмний продукт має забезпечувати можливості:

а) забезпечувати можливість завантаження текстових файлів у форматах TXT, DOCX і PDF;

б) аналізувати текст за кількома параметрами: орфографія; неоднозначні, суб’єктивні та неточні слова; неоднозначні, суб’єктивні та неточні речення. Формулювання є нечітким, якщо його не можливо точно виміряти. Формулювання є неоднозначним, якщо воно має більше одного трактувань. Формулювання є суб’єктивним, якщо його оцінка залежить від суб’єкту, що оцінює;

в) відображати результати аналізу у вигляді звіту з зазначенням типу помилок, їх розташування та можливих виправлень.

3.1.4. Вхідні дані

Текстові документи, що містять вимоги до веб-сайту, представлені у форматах TXT, DOCX, PDF, або набрані вручну у спеціальній текстовій формі. Дані можуть включати функціональні, нефункціональні та бізнес-вимоги.

3.1.5. Вихідні дані

Вихідні дані включають:

а) звіт про аналіз вимог у текстовому форматі;

б) інтерактивний візуальний звіт в графічному інтерфейсі користувача.

3.1.6. Опис зовнішнього інформаційного середовища

Система працює у середовищі операційних систем Windows. Для роботи програми необхідно встановлення бібліотек Python, таких як tkinter, language_tool_python, spacy, та інших, залежно від специфікації. Інтерактивний інтерфейс реалізований із використанням бібліотеки customtkinter.

3.2. Внутрішнє проектування

3.2.1. Проектування інтерфейсу користувача

Програма реалізує графічний інтерфейс користувача (GUI). GUI можна умовно поділити на три зони:

а) menu;

б) text;

в) mistakes.

Схему графічного інтерфейсу користувача було наведено у рисунку 3.1.

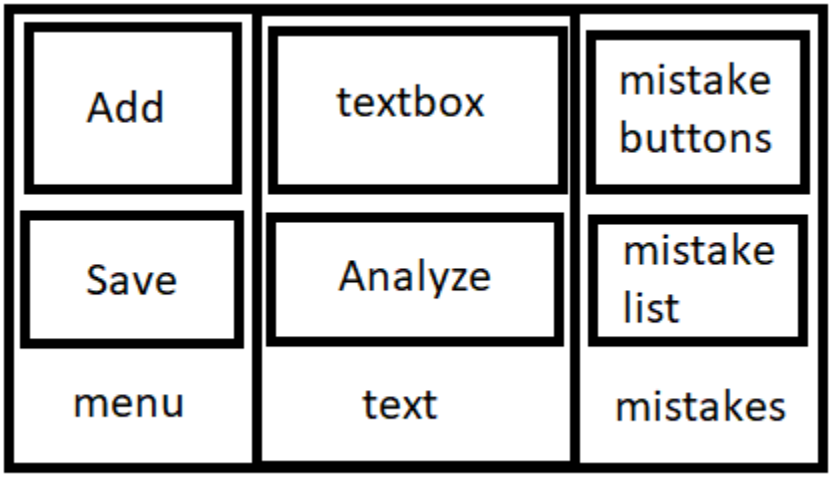

Рисунок 3.1 – схема графічно інтерфейсу користувача

У menu відбувається вхід і вихід даних за допомогою кнопок керування Add та Save. У Text зоні відбувається робота безпосередньо з вимогами до сайту. Textbox надає можливість переглянути та відредагувати текст. За допомогою кнопки Analyze виконується аналіз введеного тексту. У Mistakes зоні відображається результат аналізу тексту вимог. За допомогою Mistake Buttons користувач обирає тип помилок, який його цікавить, за допомогою Mistake List користувач переглядає помилки.

3.2.2. Реалізація інтерфейсу користувача

Реалізація інтерфейсу користувача базується на його простоті, зручності у використанні та інтерактивності. Інтерфейс реалізовано за допомогою бібліотеки customtkinter, яка дозволяє створювати сучасні, налаштовані графічні інтерфейси на основі класичної бібліотеки tkinter. Цей підхід забезпечує баланс між простотою коду та функціональністю.

Інтерфейс поділено на три основні фрейми: меню для керування файлами, текстове поле для відображення вмісту та аналізу, і панель результатів для відображення помилок і їх опису. Зовнішній вигляд форми програми згідно інтерфейсу було наведено у рисунку 3.2.

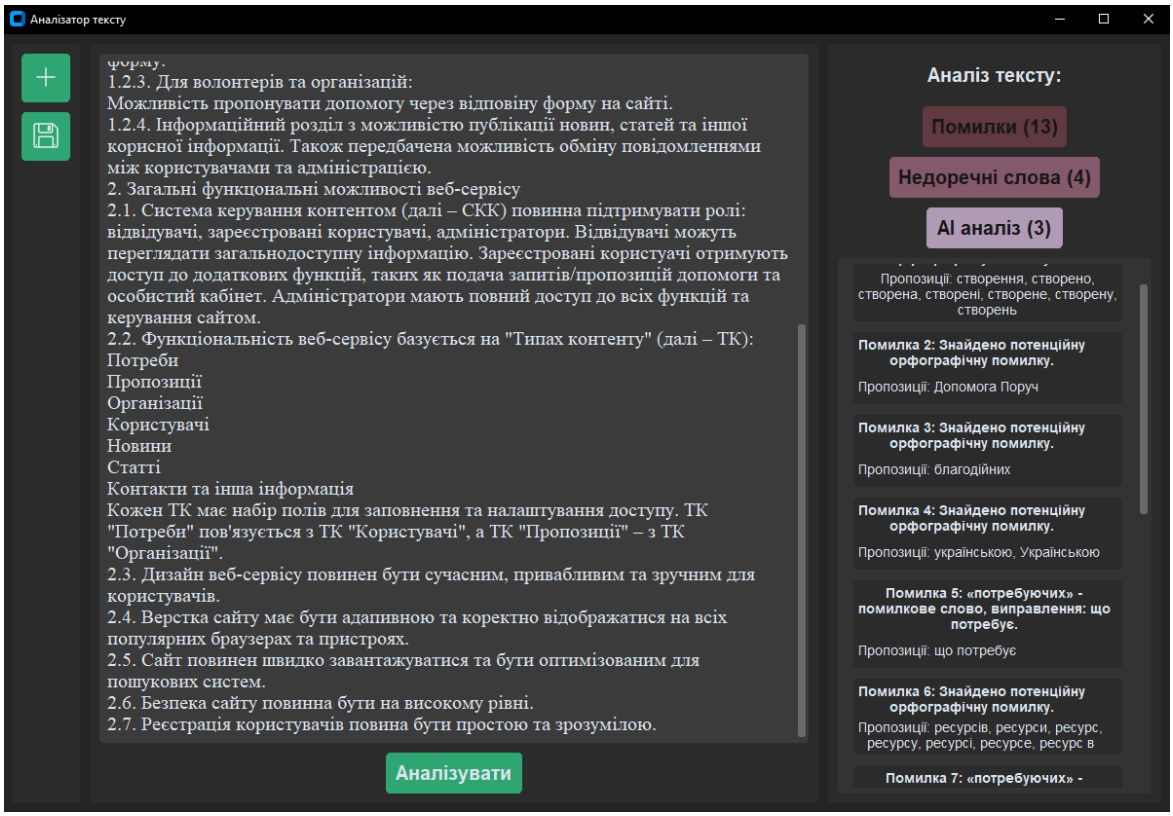

Рисунок 3.2 – Зовнішній вигляд форми програми

Кожен із цих фреймів виконує чітко визначену функцію, що робить інтерфейс інтуїтивно зрозумілим.

Для завантаження та збереження файлів реалізовано інтерактивні кнопки з іконками. Це забезпечує швидкий доступ до основних функцій. Текстове поле підтримує редагування тексту, автоматичне оновлення вмісту та виділення помилок. Кнопки для аналізу тексту дозволяють виконувати завдання одним кліком, після чого результати одразу відображаються у відповідних секціях.

Інтерфейс підтримує зміну розмірів вікна, і всі компоненти автоматично налаштовуються під доступний простір. Це дозволяє ефективно використовувати екран незалежно від його роздільної здатності.

Використання customtkinter дозволяє швидко створювати стильні та функціональні елементи, зменшуючи обсяг коду порівняно з ручним налаштуванням компонентів.

Після запуску програми користувач бачить чіткий та організований інтерфейс. Усе необхідне для роботи доступно в один-два кліки. Наприклад:\n- Завантаження файлу автоматично викликає функцію для відображення його вмісту в текстовому полі. Після натискання кнопки аналізу програма запускає обробку тексту, а результати одразу з’являються у відповідному вікні. Користувач може натиснути на будь-який елемент списку помилок, щоб побачити виділене місце помилки в тексті. Демонстрацію виділення тексту було наведено у рисунку 3.3.

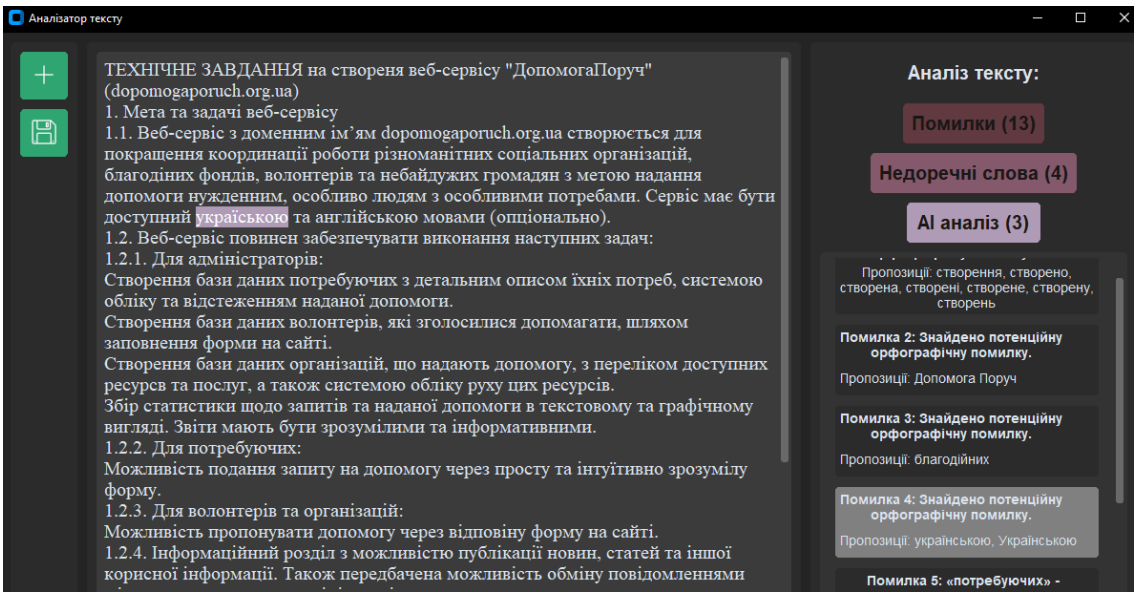

Рисунок 3.3 – Демонстрація виділення помилки у тексті

Таким чином, інтерфейс програми реалізований так, щоб забезпечити максимальну зручність і ефективність для користувача, при цьому залишаючись простим для підтримки та модернізації. Даний інтерфейс напряму реалізує функціональні вимоги програми, а саме відображення результатів аналізу у вигляді звіту з зазначенням типу помилок, їх розташування та можливих виправлень, та взаємодії через UI.

3.2.3. Проектування логіки системи

Логіка системи основана за принципом реагування на події, що означає, що програма виконує певні дії у відповідь на дії користувача. Цей підхід забезпечує інтерактивність програми.

В контексті програми, подія – це будь-яка дія користувача, яка призводить до зміни стану програми або до виконання певних операцій. До основних подій, на які реагує програма, належать: завантаження файлу, натискання кнопки аналізу та збереження файлу. Логіку системи можна представити у вигляді діаграми прецедентів, наведеної на рисунку 3.4.

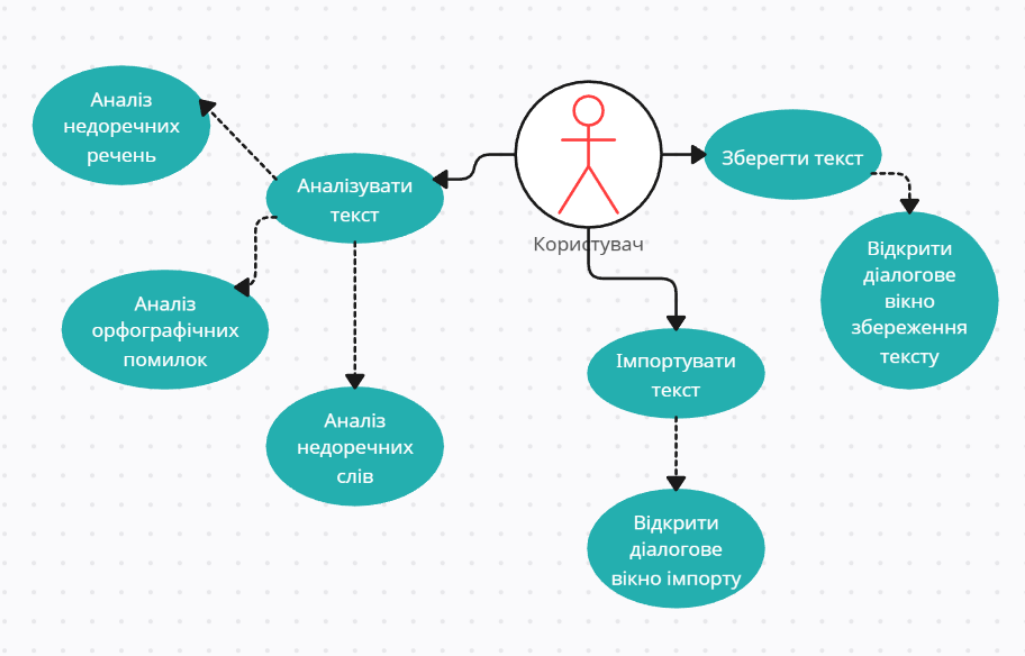

Рисунок 3.4 – Діаграма прецедентів

3.2.4. Реалізації методів верифікації вимог до сайту

Було реалізовано дві основні функції у програмі:

а) запис та збереження вхідних даних;

б) обробка та верифікація вимог до сайту шляхом аналізу тексту.

Запис вхідних даних та їх збереження відбувається за допомогою функція open_file() та save_file(), які відповідно викликаються кнопками add_button та

save_button. Щоб виконати читання docx та pdf файлів були створені алгоритми та описані у функції read_file(). Перелік функцій у коді було наведено у рисунку 3.5.

Рисунок 3.5 – функції запису та збереження вхідних даних

Обробка та верифікація вимог до сайту відбувається за допомогою трьох функцій обробки, кожна з яких аналізує текст за власним алгоритмом. За верифікацію вимог до сайту уповноважені три функції: check_mistakes(), check_miswords(), check_misphrases(). Кожна функція відповідно аналізує текст на помилки; на кількість неточних, неоднозначних, суб'єктивно-забарвлених слів; на кількість невлучно створених речень. Перелік функцій обробки тексту у коді було наведено у рисунку 3.6.

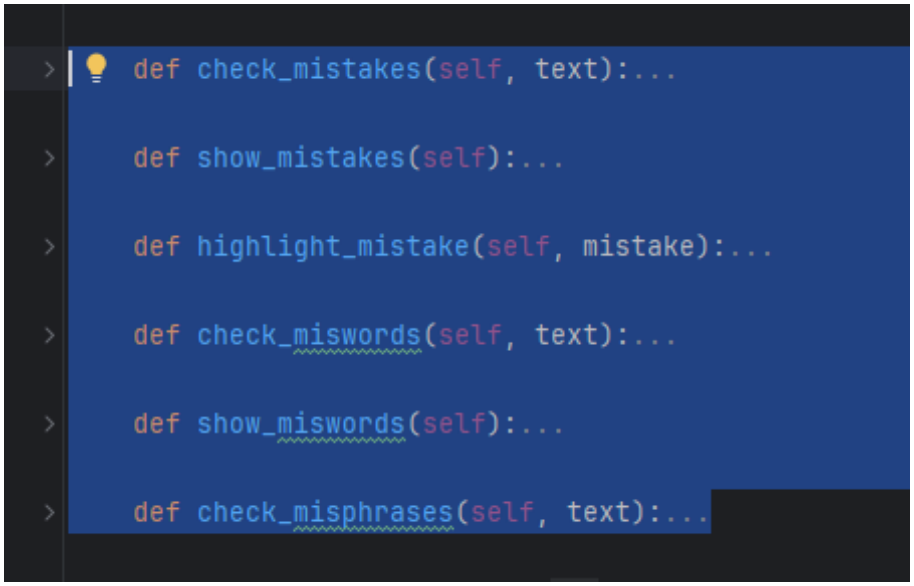

Рисунок 3.6 – функції обробки та верифікації вимог до сайту

3.2.4.1. Пошук орфографічних помилок

Пошук орфографічних помилок було розроблено за допомогою LanguageTool API для Python. LanguageTool показав високі результати пошуку орфографічних помилок на стадії дослідження (див. Розділ 2). Цей інструмент підтримує перевірку текстів багатьма мовами, включаючи українську, і забезпечує виявлення орфографічних, граматичних та стилістичних помилок. Пошук орфографічних помилок поділяється на такі етапи:

а) попередня обробка тексту — користувач завантажує текстовий документ у програму або вводить текст у спеціальне текстове поле. Програма перевіряє текст на наявність символів або форматування, яке може заважати аналізу (наприклад, приховані символи, зайві пробіли тощо). Якщо текст отримано з файлу, він спочатку конвертується у зручний для аналізу формат;

б) передача тексту до LanguageTool — оброблений текст передається у вигляді рядка до методу check() бібліотеки language_tool_python. LanguageTool аналізує текст, використовуючи свій вбудований словник і набір граматичних правил, щоб виявити орфографічні помилки;

в) виявлення помилок — для кожного знайденого відхилення LanguageTool створює об'єкт помилки (Match);

г) фільтрація — програма фільтрує помилки, щоб виключити ті, які не є релевантними або корисними для аналізу;

д) генерація результатів — для кожної помилки зберігається інформація. Усі знайдені помилки додаються до списку для подальшого відображення у графічному інтерфейсі.

Код обробки офрографічних помилок було наведено у Додатку Б. Дана функція напряму реалізує функціональні вимоги програми, а саме аналіз тексту на орфографічні помилки.

3.2.4.2. Пошук недоречних слів

Пошук недоречних слів реалізовано за допомогою Python бібліотеки SpaCy для обробки природної мови. SpaCy показав певні результати під час дослідження (див. Розділ 2). У програмі використовується попередньо визначений список категорій слів, які можуть бути потенційно недоречними в текстах вимог до сайтів. Такий підхід дозволяє виявляти слова або вирази, які можуть викликати неоднозначність, є занадто загальними чи не додають цінності до тексту. Пошук недоречних слів поділяється на такі етапи:

а) попередня обробка тексту — користувач завантажує текстовий документ або вводить текст у програму. Текст проходить базову обробку: видаляються зайві пробіли, символи, які не несуть значення, і виконується сегментація на речення та токени (окремі слова);

б) ініціалізація мовної моделі — для аналізу тексту використовується модель uk_core_news_sm із бібліотеки spacy, яка підтримує українську мову. Модель завантажується для лемматизації (перетворення слова в його базову форму) та аналізу тексту.

в) визначення категорій недоречних слів — у програмі визначено кілька категорій слів, які перевіряються під час аналізу. Такі категорії як неоднозначність, неточність та суб’єктивність;

г) аналіз тексту — кожне слово з тексту порівнюється зі списком слів із зазначених категорій. Під час аналізу перевіряється як лемма слова (базова форма), так і його контекст. Це дозволяє виявляти різні форми одного й того самого слова (наприклад, "зручний" і "зручніше");

д) генерація результатів — для кожного слова зберігається інформація. Усі знайдені слова додаються до списку для подальшого відображення у графічному інтерфейсі.

Код обробки недоречних слів було наведено у Додатку Б. Дана функція напряму реалізує функціональні вимоги програми, а саме аналіз тексту на неточності, неоднозначності та суб’єктивні оцінки.

3.2.4.3. Пошук недоречних речень

Пошук недоречних речень виконується за допомогою аналізу тексту з використанням API OpenAI (GPT) та бібліотеки nltk. Алгоритм аналізує текстові вимоги і визначає, які речення не відповідають стандартам чіткості, специфічності та релевантності до теми тексту, а саме вимоги до сайту.

GPT (Generative Pre-trained Transformer) — це сучасна модель обробки природної мови, розроблена компанією OpenAI. Вона здатна генерувати текст, розуміти контекст і виконувати завдання аналізу текстів завдяки своїй потужній архітектурі, яка базується на трансформерах. GPT навчається на великих масивах текстових даних, що дозволяє їй аналізувати синтаксис і семантику тексту, розуміти нюанси мови, включаючи складні конструкції та ідіоми, пропонувати обґрунтовані виправлення та вдосконалення тексту.

У цьому проекті GPT використовується через OpenAI API, що забезпечує зручний спосіб взаємодії з моделлю. Під час аналізу тексту GPT оцінює кожне речення на основі заданих критеріїв, таких як релевантність, чіткість і специфічність, і пропонує рекомендації щодо виправлення недоліків.

NLTK (Natural Language Toolkit) — це одна з найпопулярніших бібліотек для роботи з текстами у Python. Вона надає широкий набір інструментів для обробки природної мови, включаючи токенізацію (розділення тексту на речення або слова), лемматизацію (перетворення слова в його базову форму), видалення стоп-слів і обробку текстів.

У нашій програмі NLTK використовується для токенізації тексту на речення за допомогою функції sent_tokenize(). Це забезпечує чіткий розподіл тексту на логічні частини для подальшого аналізу за допомогою GPT.

Пошук недоречних речень поділяється на такі етапи:

а) попередня обробка тексту — текст вимог завантажується користувачем через інтерфейс програми. Програма видаляє зайві символи та розбиває текст на окремі речення за допомогою бібліотеки nltk (метод sent_tokenize). Кожне речення зберігається разом із його позицією у вихідному тексті для подальшого виділення помилок;

б) визначення критеріїв аналізу — для виявлення недоречних речень використовуються певні критерії. Чіткість: речення не повинні містити нечітких або абстрактних виразів, таких як «дуже зручний» чи «швидко працюючий». Специфічність: у реченні мають бути конкретні вимоги, наприклад, «Сторінка повинна завантажуватися за 2 секунди». Релевантність: речення має відповідати темі «Вимоги до веб-сайту» і не містити інформації, яка не стосується предмета вимог. Доречність: речення не повинні містити зайвих слів чи неконкретних фраз, які ускладнюють розуміння;

в) аналіз за допомогою OpenAI API — кожне речення передається до моделі GPT через API OpenAI. Для аналізу використовується історія повідомлень (prompt), яка містить опис критеріїв, приклади недоречних речень і правильних формулювань. GPT аналізує кожне речення;

г) генерація результатів — для кожного недоречного речення зберігається інформація. Усі знайдені слова додаються до списку для подальшого відображення у графічному інтерфейсі.

Код обробки недоречних речень було наведено у Додатку (?). Дана функція напряму реалізує функціональні вимоги програми, а саме аналіз

тексту на неточності, неоднозначності та суб’єктивні оцінки, але на більш глибинному рівні.

3.3. Обґрунтування вибору мови програмування Python

Python був обраний як основна мова програмування для розробки цієї програми з огляду на його численні переваги, які сприяють швидкому, ефективному і надійному створенню програмного забезпечення.

Мова має розгалужену екосистему бібліотек, які спрощують розробку додатків для обробки тексту, побудови графічного інтерфейсу та інтеграції з іншими системами. У цьому проекті використовувалися такі бібліотеки:

а) language_tool_python для перевірки орфографії та граматики;

б) spacy для семантичного аналізу тексту;

в) nltk для токенізації та інших базових операцій з текстом;

г) tkinter та customtkinter для створення сучасного графічного інтерфейсу;

д) docx та pypdf для роботи з текстовими файлами у форматах DOCX і PDF.

Дана мова програмування відома своїм лаконічним і зрозумілим синтаксисом, який дозволяє розробникам швидко створювати й підтримувати програми. Це особливо важливо для складних проектів, які потребують регулярного оновлення чи масштабування.

Python став оптимальним вибором для реалізації програми завдяки своїй гнучкості, великій кількості спеціалізованих бібліотек, підтримці сучасних інструментів і простоті використання. Це дозволило створити багатофункціональну програму з ефективним графічним інтерфейсом і потужними можливостями аналізу текстів у стислі терміни.

3.4. Опис роботи програми

У цьому підрозділі наведено детальний опис роботи програми, процес імпорту текстового файлу, його обробки та експорту.

Спочатку програму треба запустити. Запуск програми виконується за допомого подвійного натискання лівою кнопкою миші на виконуваному файлі .exe. Після цього користувач потрапляє у головне меню програми, яке наведено на рисунку 3.7.

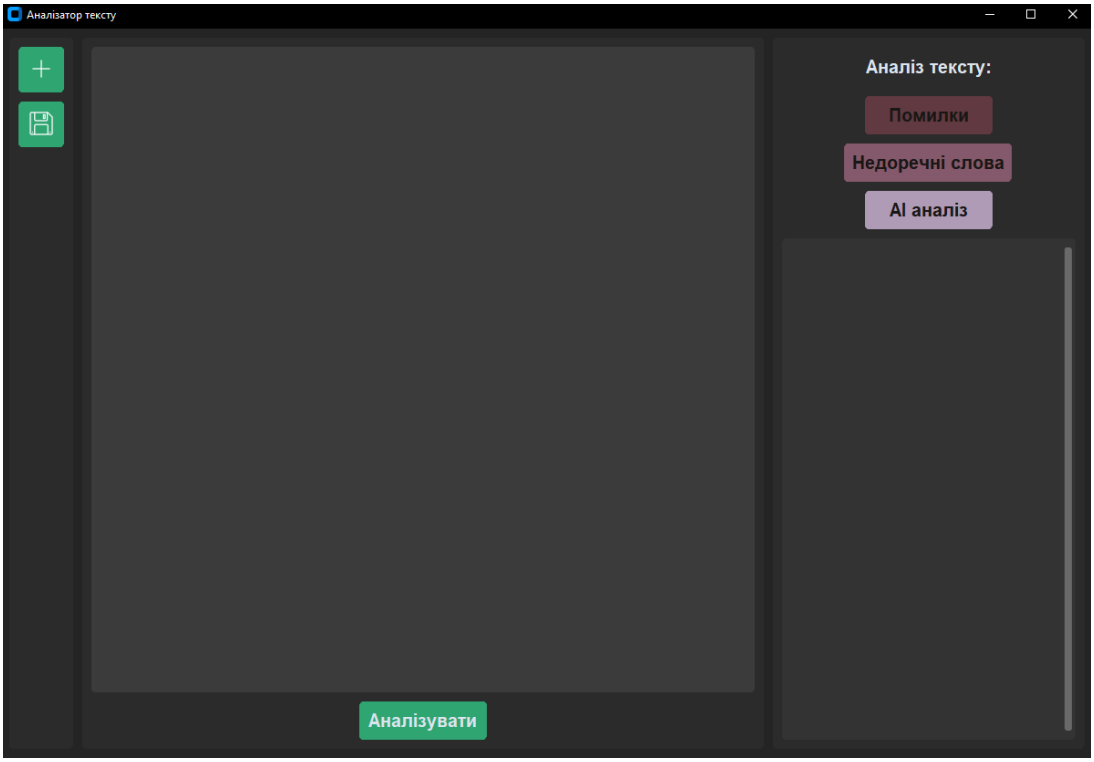

Рисунок 3.7 – Головне меню програми

Наразі користувач не бачить вимог до сайту та знайдених помилок. Щоб почати роботу у застосунку користувач повинен створити текст у текстовому полі посередині, або імпортувати вже існуючий текст. Імпортуємо текст за допомогою кнопки «+» у верхньому лівому куті. Після натискання на кнопку користувач потрапляє у діалогове вікно, яке наведено на рисунку 3.8. Користувач може обирати між типами обираємого файлу, а саме txt, docx, pdf, що відповідає функціональним вимогам програми.

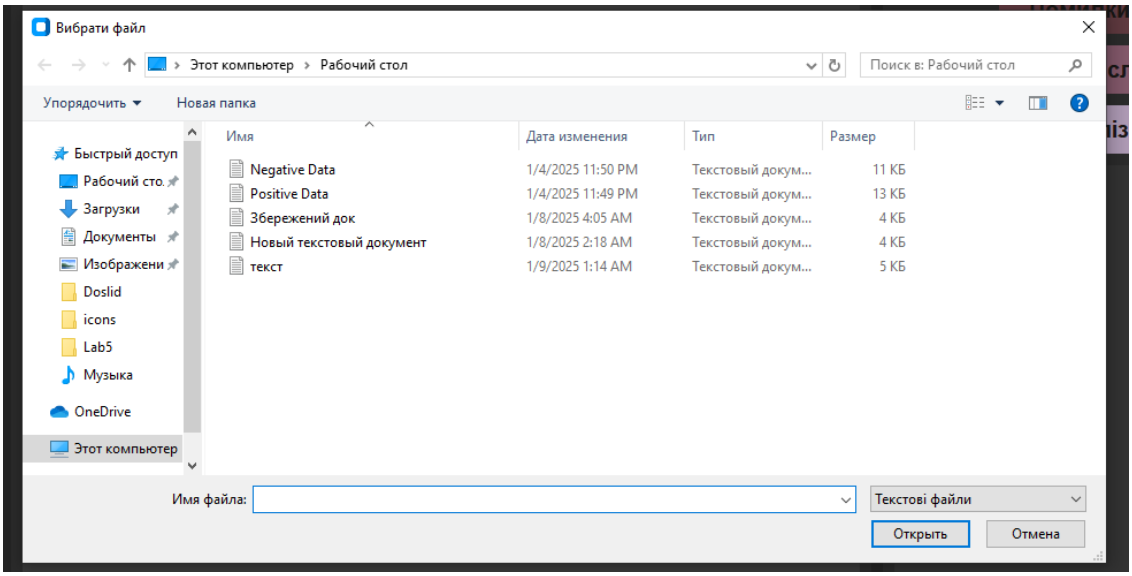

Рисунок 3.8 – Діалогове вікно зчитування вимог

Оберемо файл, після чого отримаємо результат у вигляді заповненого текстового поля. Текст можна додаткового відкоригувати у текстовому полі. Після завершення всіх змін користувач натискає кнопку «Аналізувати». Після певного проміжку часу користувач фіксує зміни – на кнопках «Помилки», «Недоречні слова» та «AI аналіз» з’явились числа, що вказують на кількість знайдених недоліків. Дані зміни наведено у рисунку 3.9.

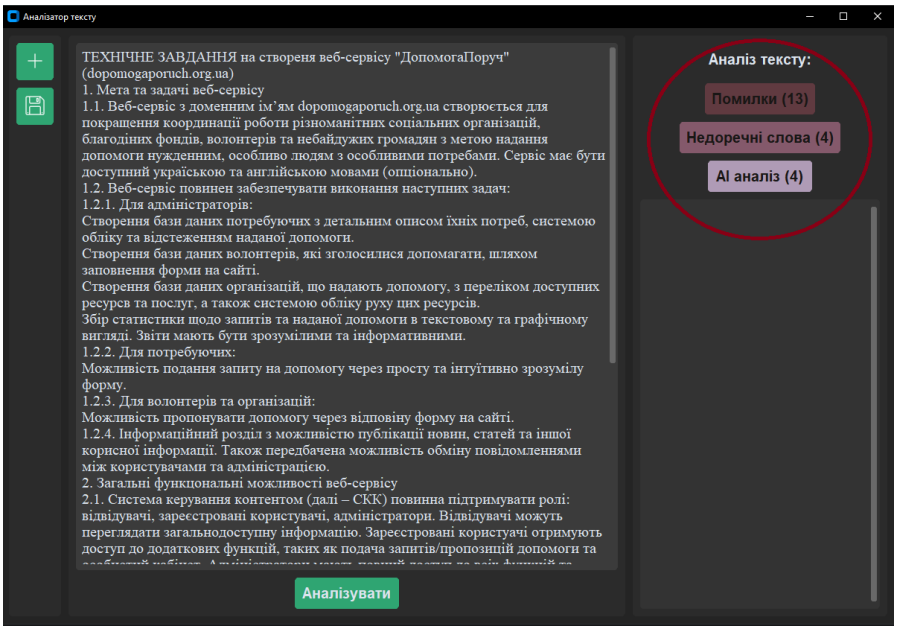

Рисунок 3.9 – Результат аналізу вимог до сайту

Щоб подивитись конкретні недоліки, користувач натискає кнопку зацікавленого недоліку. При натисканні кнопки «Помилки (13)» ми побачимо список помилок, які слід усунути у вимогах до сайту. Натискання на кожний недолік посилає нас на конкретне слово у текстовому полі и виділяє це слово кольором. Дану роботу було ілюстровано раніше, див. рис. 3.3.

Аналогічна робота прослідковується з двома іншими кнопками. При натисканні кнопки «Недоречні слова (4)» користувач побачить список недоречних слів, а при натисканні «AI аналіз (4)» — список недоречних речень, що зображено на рисунку 3.10.

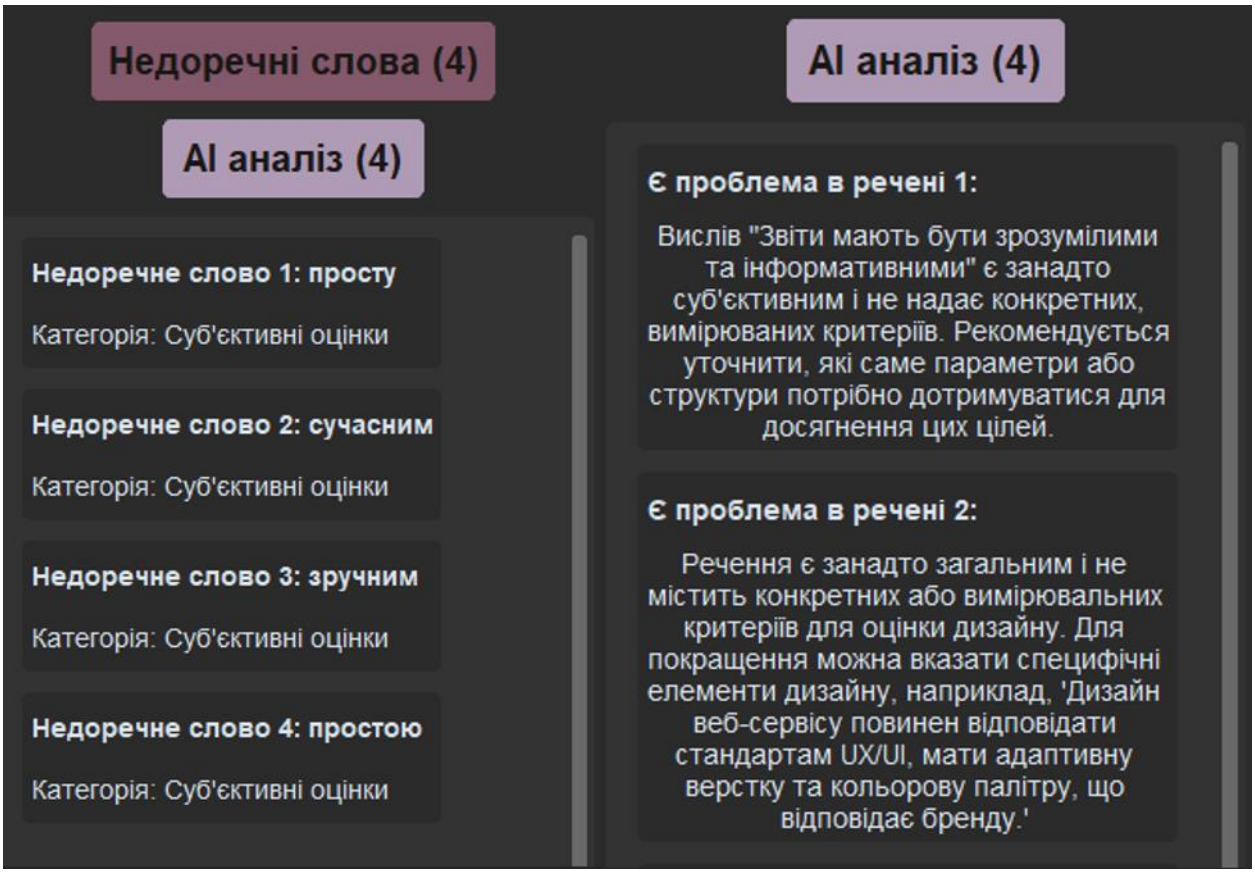

Рисунок 3.10 – Списки недоречних слів та речень

Після того, як користувач побачить недоліки, він їх усунить у текстовому полі. Щоб зберегти зміни натиснемо кнопку «Зберегти», яку розташовано під кнопкою «Додати». Користувач потрапляє у діалогове вікно, де він може зберегти свої зміни, що проілюстровано у рисунку 3.11.

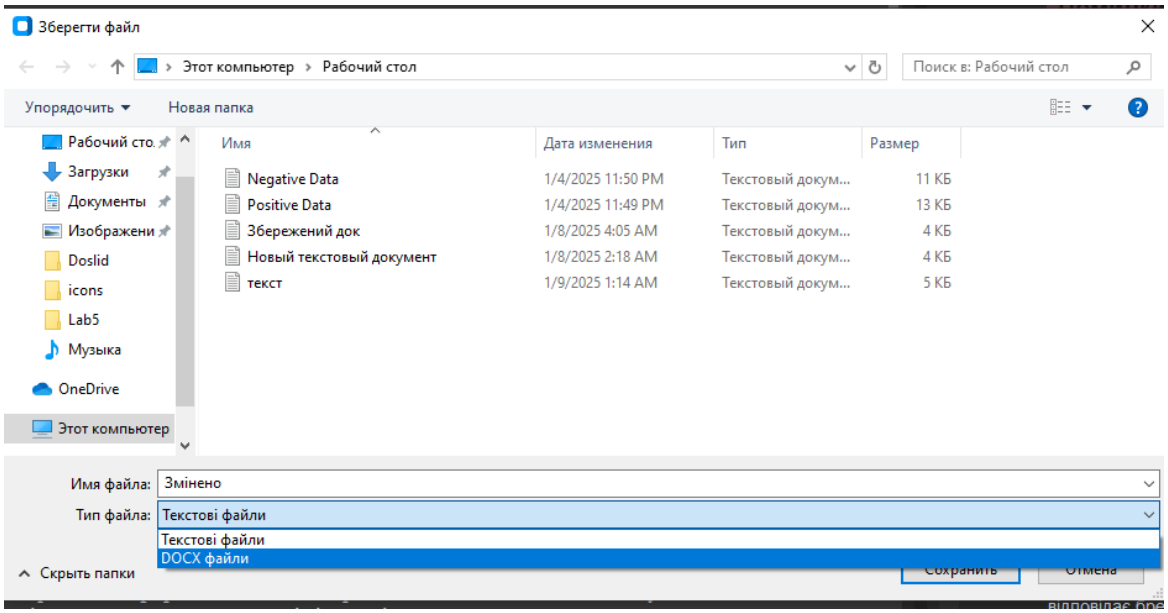

Рисунок 3.11 – Діалогове вікно збереження вимог до сайту

Зберегти дані можна у текстовому форматі, або у форматі docx для подальшого користування. Як бачимо, програмне забезпечення працює без помилок, а інтерфейс користувача відповідає функціональним вимогам.

Висновки до розділу 3

У цьому розділі було детально описано процес проектування та розробки програмного забезпечення для автоматичної верифікації вимог до веб-сайтів. Розглянуто зовнішнє та внутрішнє проектування програми, особливості інтерфейсу користувача, а також алгоритми верифікації вимог. Було обґрунтовано вибір мови програмування Python та описано використані бібліотеки. Крім того, було проведено тестування програми для перевірки її функціональності та відповідності вимогам.

Проведене тестування програми показало її працездатність та відповідність функціональним вимогам. Описано процес запуску програми, імпорту тексту, аналізу та перегляду результатів. Тестування підтвердило, що програма коректно обробляє текстові файли різних форматів, відображає знайдені помилки у зручному форматі та забезпечує зручну навігацію по

тексту. Результати тестування свідчать про те, що інтерфейс програми є інтуїтивно зрозумілим та зручним для користувача.

РОЗДІЛ 4. ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПРОГРАМИ

Цей розділ присвячено дослідженню ефективності розробленої програми для автоматичної верифікації вимог до веб-сайтів. Метою дослідження є оцінка здатності програми виявляти різні типи помилок у текстах вимог та порівняння її ефективності з результатами, отриманими в розділі 2 за допомогою ручного аналізу, LanguageTool та NLP-аналізатора (spaCy Python).

Дослідження проводилося шляхом аналізу того ж тексту вимог, що використовувався в розділі 2 (описаний в підпункті 2.2.2), за допомогою розробленої програми. Результати аналізу порівнювалися з результатами попередніх досліджень для визначення переваг та недоліків програми.

4.1. Процедура проведення дослідження

Дане дослідження використовує показник та вхідні данні з дослідження з розділу 2. Процедура дослідження включає наступні кроки:

а) запуск програми — програма запускалася на комп'ютері з операційною системою Windows згідно з інструкцією, описаною в розділі 3.4;

б) завантаження тексту вимог — текст вимог, що використовувався в розділі 2, був завантажений в програму у форматі TXT;

в) аналіз тексту — текст було проаналізовано за допомогою функцій програми, що відповідають за виявлення орфографічних помилок, неточностей, неоднозначностей та суб'єктивних оцінок;

г) збір результатів — результати аналізу, що відображалися у звіті програми, були задокументовані. Звіт містив інформацію про кількість та типи знайдених помилок, а також їх розташування в тексті;

д) порівняння результатів — результати, отримані за допомогою програми, були порівняні з результатами ручного аналізу, LanguageTool та NLP-аналізатора з розділу 2.

4.2. Результати дослідження

Для верифікації тексту вимог за допомогою створеної програми, текст було завантажено в програму, та отримані результати було задокументовано, відповідно до таблиці 4.1.

Таблиця 4.1 — Результат верифікації створеною програмою

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Кількість	11	3	2	6	1

4.3. Порівняння результатів з дослідженням з розділу 2

Результати, отримані за допомогою розробленої програми, було порівняно з результатами ручного аналізу, LanguageTool та NLP-аналізатора з розділу 2 та занесено до таблиці 4.2:

Таблиця 4.2 – Результат верифікації всіх методів

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Ручний метод	9,5	3,125	2	5	26,125
LanguageTool	11	0	0	0	1
NLP-аналізатор (spaCy Python)	0	3	2	5	1

	Орфографічна помилка, $N_{орф}$	Неточність, $N_{неточн}$	Неоднозначність, $N_{неодн}$	Суб'єктивна оцінка, $N_{суб}$	$t_{сер}$ (хв)
Програма	11	3	2	6	1

Порівняння методів за орфографічною помилкою представлено у вигляді діаграми, рисунок 4.1:

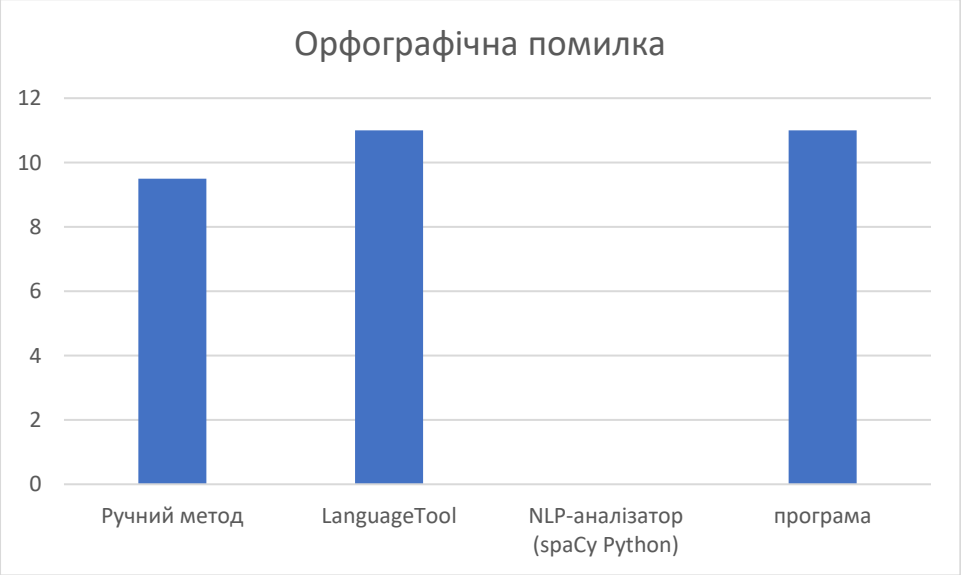

Рисунок 4.1 — Порівняння методів за орфографічною помилкою

Порівняння методів за помилкою неточності представлено у вигляді діаграми, рисунок 4.2:

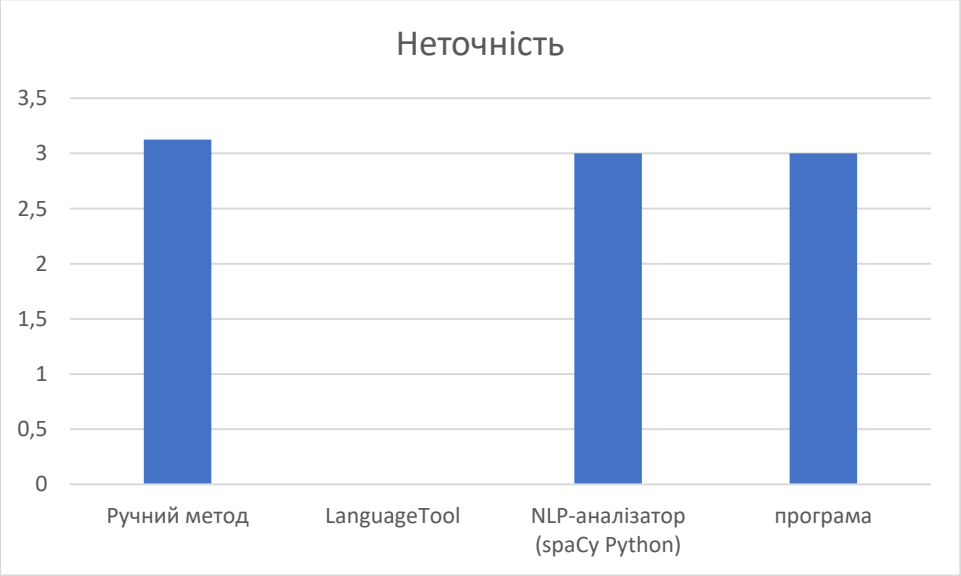

Рисунок 4.2 — Порівняння методів за помилкою неточності

Порівняння методів за помилкою неоднозначності представлено у вигляді діаграми, рисунок 4.3:

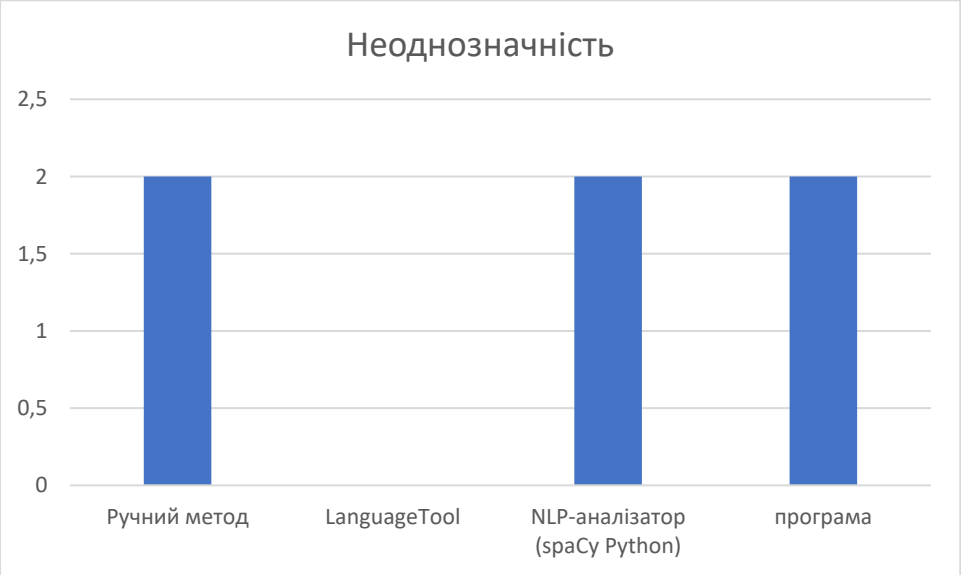

Рисунок 4.3 — Порівняння методів за помилкою неоднозначності

Порівняння методів за суб’єктивною оцінкою представлено у вигляді діаграми, рисунок 4.4:

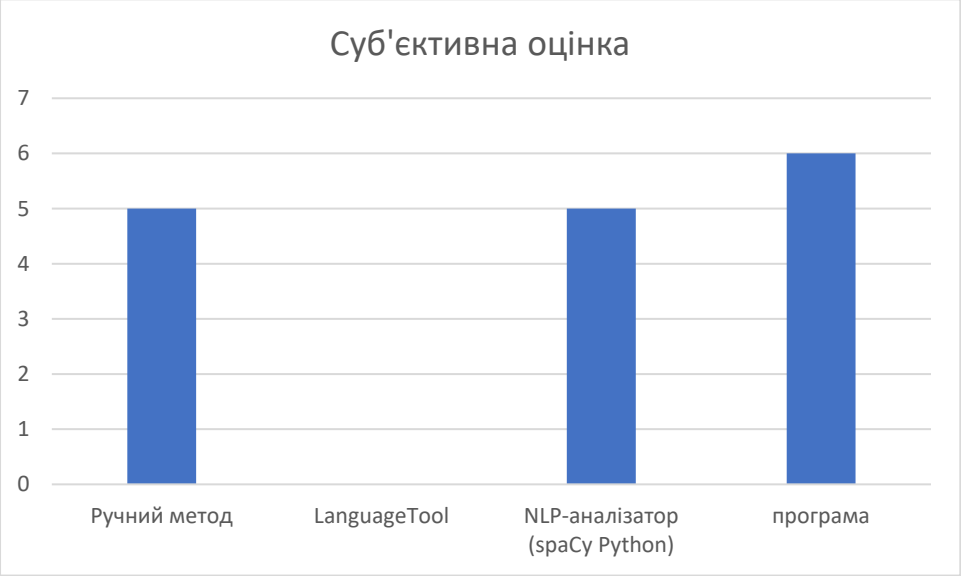

Рисунок 4.4 — Порівняння методів за суб’єктивною оцінкою

Порівняння методів за часом роботи представлено у вигляді діаграми, рисунок 4.5:

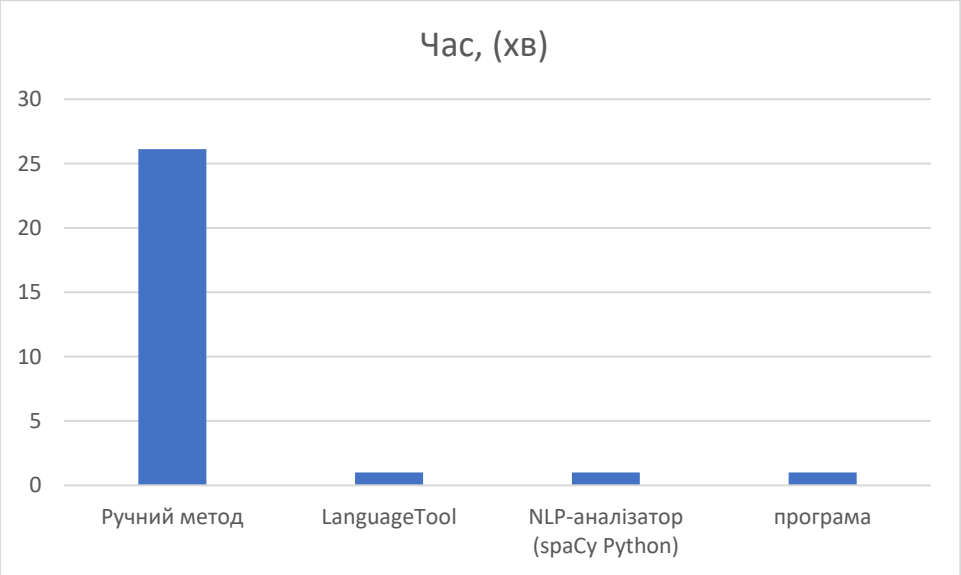

Рисунок 4.5 — Порівняння методів за часом роботи

4.4. Аналіз результатів

Порівняння результатів показує, що розроблена програма демонструє ефективність, близьку до ефективності NLP-аналізатора з розділу 2, у виявленні неточностей, неоднозначностей та суб'єктивних оцінок. Щодо

виявлення орфографічних помилок, програма показує результат, аналогічний LanguageTool.

Висновки до розділу 4

У цьому розділі було проведено дослідження ефективності розробленої програми для автоматичної верифікації вимог до веб-сайтів шляхом порівняння її результатів з результатами ручного аналізу, LanguageTool та NLP-аналізатора, представленими в розділі 2. Аналіз показав, що програма успішно реалізує поставлені завдання, демонструючи ефективність, близьку до NLP-аналізатора, у виявленні неточностей, неоднозначностей та суб'єктивних оцінок. Щодо орфографічних помилок, програма досягла результатів, аналогічних до LanguageTool. Це свідчить про те, що розроблений інструмент вдало поєднує переваги автоматизованих методів, забезпечуючи комплексний підхід до верифікації вимог. Результати дослідження підкреслюють доцільність використання автоматизованих методів для підвищення ефективності та об'єктивності процесу верифікації, а також підтверджують практичну цінність розробленої програми як корисного інструменту для аналітиків, тестувальників та розробників, що працюють з вимогами до веб-сайтів.

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

У цій роботі було досліджено методи верифікації вимог до веб-сайтів та розроблено програмне забезпечення для автоматизації цього процесу. Проведено аналіз існуючих методів верифікації, включаючи ручний аналіз та автоматизовані підходи з використанням LanguageTool та NLP-аналізатора на базі spaCy. Експериментальне дослідження показало, що кожен метод має свої сильні та слабкі сторони. Ручний аналіз, хоча й забезпечує глибоке розуміння контексту, є трудомістким та суб'єктивним. LanguageTool ефективно виявляє орфографічні та стилістичні помилки, але обмежений у виявленні семантичних проблем. Розроблений NLP-аналізатор продемонстрував хороші результати у виявленні неточностей, неоднозначностей та суб'єктивних оцінок, що підтверджує ефективність застосування методів обробки природної мови для верифікації вимог. Розроблене програмне забезпечення успішно інтегрувало переваги LanguageTool та NLP-аналізатора, забезпечуючи комплексний підхід до перевірки вимог.

Результати дослідження підтверджують гіпотезу про те, що автоматизація процесу верифікації вимог дозволяє підвищити його ефективність з точки зеконмленого часу. Розроблена програма може бути корисним інструментом для аналітиків, тестувальників та розробників, що працюють з вимогами до веб-сайтів, дозволяючи їм швидше та якісніше перевіряти тексти вимог на наявність різних типів помилок.

На основі проведеного дослідження можна сформулювати наступні рекомендації: для досягнення найкращих результатів рекомендується використовувати комбінований підхід до верифікації вимог, що поєднує ручний аналіз з автоматизованими інструментами. LanguageTool можна використовувати для швидкої перевірки орфографії та стилістики, тоді як розроблена програма або аналогічні NLP-інструменти можуть бути застосовані для виявлення семантичних проблем. Ручний аналіз залишається

важливим для перевірки контексту та складних випадків, які важко автоматизувати.

Для подальших досліджень можна рекомендувати розширення словника ключових слів для NLP-аналізатора, вдосконалення алгоритмів аналізу з використанням більш складних методів NLP, таких як аналіз залежностей між словами та семантичний аналіз тексту, а також проведення більш масштабних експериментів з залученням більшої кількості учасників та використанням різних текстів вимог. Крім того, перспективним напрямком є дослідження можливості інтеграції розробленої програми з іншими інструментами для управління вимогами, що дозволить створити комплексну систему для забезпечення якості вимог протягом усього життєвого циклу розробки програмного забезпечення.

ПЕРЕЛІК ПОСИЛАНЬ

1. Boehm, B. W. Software engineering economics / B. W. Boehm // IEEE Transactions on Software Engineering. – 1976. – Vol. SE-2, № 4. – P. 344-351
2. Confluence: офіційний сайт. URL: https://www.atlassian.com/software/confluence
3. CustomTkinter. URL: https://github.com/TomSchimansky/CustomTkinter
4. ДСТУ 4163:2020. Державна уніфікована система документації. Уніфікована система організаційно-розпорядчої документації. Вимоги до оформлення документів. Чинний від 2021-09-01. Вид. офіц. Київ : УкрНДНЦ, 2021. 37 с.
5. Hull, E., Jackson, K., Dick, J. Requirements Engineering / E. Hull, K. Jackson, J. Dick. – 2nd ed. – Springer, 2005. – 488 p.
6. Jira Software: офіційний сайт. URL: https://www.atlassian.com/software/jira
7. LanguageTool API для Python. URL: https://languagetool.org/
8. Leffingwell, D. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise / D. Leffingwell. – Addison-Wesley Professional, 2011. – 544 p.
9. NLTK (Natural Language Toolkit). URL: https://www.nltk.org/
10. OpenAI API (GPT). URL: https://platform.openai.com/docs/
11. Pressman, R. S., Maxim, B. R. Software Engineering: A Practitioner's Approach / R. S. Pressman, B. R. Maxim. – 8th ed. – McGraw-Hill Education, 2015. – 912 p.
12. Python бібліотека docx. URL: https://python-docx.readthedocs.io/
13. Python бібліотека PyPDF. URL: https://pypdf.readthedocs.io/
14. Sommerville, I. Software Engineering / I. Sommerville. – 10th ed. – Pearson Education Limited, 2016. – 816 p.
15. SpaCy для обробки тексту. URL: https://spacy.io/
16. Visual Paradigm: офіційний сайт. URL: https://www.visual-paradigm.com/
17. Wiegers, K. E., Beatty, J. Software Requirements / K. E. Wiegers, J. Beatty. – 3rd ed. – Microsoft Press, 2013. – 416 p.

ДОДАТКИ

ДОДАТОК А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
28.09.24

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01444-01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
28.09.24

Керівник розробки
Олена КУРОП’ЯТНИК
28.09.24

Виконавець

Нікіта ЛОХВИЦЬКИЙ

28.09.24

Нормоконтролер
Світлана ВОЛКОВА
28.09.24

2025

ЗАТВЕРДЖЕНО
44165850.01444-01-ЛЗ

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Технічне завдання

44165850.01444-01

Листів 12

2025

АНОТАЦІЯ

Документ 44165850.01444-01 «Автоматизований аналізатор тексту вимог до сайтів. Технічне завдання» входить до складу програмної документації на програму для автоматизації пошуку і видачі на екран помилкових даних.

У даному документі представлено технічне завдання на розробку програмного застосунку.

ЗМІСТ

ВСТУП

Автоматизований аналізатор тексту вимог до сайтів призначена для автоматизації пошуку і видачі на екран помилкових даних.

Причиною виникнення продукту є те, що аналогів даному продукту не існує.

Область застосування – етап збору та аналізу вимог при розробці веб-сайтів.

ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є наказ №1187 ст. від 29.12.2023 про призначення керівників та затвердження тем кваліфікаційних робіт ОС Магістр.

ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення: програма призначена для автоматичної верифікації вимог до веб-сайтів у вигляді тексту. Основні функції включають:

а) виявлення орфографічних і граматичних помилок;

б) пошук неточних формулювань, неоднозначностей, суб’єктивних оцінок;

в) генерація звітів із виявленими помилками;

г) візуалізація аналізу за допомогою інтерактивного графічного інтерфейсу користувача (GUI).

Експлуатаційне призначення: система створена для використання аналітиками, тестувальниками та розробниками, які працюють із вимогами до сайтів. Програма забезпечує автоматизацію процесу перевірки вимог, підвищуючи ефективність і якість аналізу за рахунок зекономленого часу.

ВИМОГИ ДО ПРОГРАМИ АБО ПРОГРАМНОГО ПРОДУКТУ

4.1. Вимоги до функціональних характеристик

Вимоги до функціональних характеристик наступні:

а) завантаження текстових файлів у форматах TXT, DOCX і PDF;

б) аналізувати текст за кількома параметрами: орфографія; неоднозначні, суб’єктивні та неточні слова; неоднозначні, суб’єктивні та неточні речення. Формулювання є нечітким, якщо його не можливо точно виміряти. Формулювання є неоднозначним, якщо воно має більше одного трактувань. Формулювання є суб’єктивним, якщо його оцінка залежить від суб’єкту, що оцінює;

в) відображати результати аналізу у вигляді звіту з зазначенням типу помилок, їх розташування та можливих виправлень.

4.2. Вимоги до надійності

Вимоги до надійності наступні:

а) наявність архівної копії тексту програми на зовнішньому носії;

б) забезпечення стійкого функціонування програми;

в) контроль вхідної і вихідної інформації;

г) захист від копіювання ПЗ.

4.3. Умови експлуатації

Вимоги до кліматичних умов: температура – 21-25 °С, відносна вологість 40-60%.

Обслуговування не потрібне.

Для роботи із ПЗ достатньо однієї людини, що має досвід роботи із ПК та ознайомлена із керівництвом користувача.

4.4. Вимоги до складу і параметрів технічних засобів

Склад технічних засобів:

а) процесор з тактовою частотою 1 ГГЦ або вище;

б) доступ до мережі Інтернет;

в) 73 мб. місця на накопичувачі;

г) 2000 мб. оперативної пам’яті.

4.5. Вимоги до інформаційної і програмної сумісності

Програма має функціонувати під управлінням ОС Windows 10\11.

Програма реалізована мовою програмування Python, яка має декілька переваг. Однією з них є наявність мовних бібліотек обробки тексту, що є необхідною умовою для вирішення технічного завдання.

4.6. Вимоги до маркування і упаковки

Програма має маркуватися наступним чином: назва, розробник, контакти. Приклад маркування приведений на рис.1.

Автоматизований аналізатор тексту вимог до сайтів
Розробник: Лохвицький Н.С., 962м група
УДУНТ, кафедра КІТ
2024

Рис.1. Приклад маркування.

Вимоги до упаковки: упаковка має захищати від електромагнітних впливів та забруднення.

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу програмної документації мають входити:

а) технічне завдання

б) текст програми;

в) керівництво користувача.

Програмна документація повинна відповідати вимогам ДСТУ [1].

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Етапи розробки та терміни виконання приведені у табл. 6.1.

Таблиця 6.1 – Етапи розробки та терміни виконання

№	Етапи розробки	Терміни виконання
1	Постановка задачі	01.09.24 – 07.09.24
2	Визначення вимог до програми	08.09.24 – 14.09.24
3	Узгодження та затвердження технічного завдання	15.09.24 – 28.09.24
4	Програмування та налагодження програми	29.09.24 – 25.11.24
5	Тестування	26.11.24 – 17.12.24
6	Розробка, узгодження та затвердження програмної документації	18.12.24 – 28.12.24

ПОРЯДОК КОНТРОЛЮ І ПРИЙМАННЯ

Проводиться перевірка коректного виконання програмою закладених у ній функцій, тобто здійснюється функціональне тестування програми. Також здійснюється візуальна перевірка інтерфейсу програми. Строки проведення випробувань обговорюються додатково.

Контроль виконання здійснює керівник розробки.

Приймання виконується комісією.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. — 38 с.

ДОДАТОК Б

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
28.12.24

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01444-01 12 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
28.12.24

Керівник розробки
Олена КУРОП’ЯТНИК
28.12.24

Виконавець

Нікіта ЛОХВИЦЬКИЙ

28.12.24

Нормоконтролер
Світлана ВОЛКОВА
28.12.24

2025

ЗАТВЕРДЖЕНО
44165850.01444-01 12 01-ЛЗ

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Текст програми
44165850.01444-01 12 01

Листів 27

2025

АНОТАЦІЯ

Документ 44165850.01444-01 12 01 “АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ. Текст програми” входить до складу програмної документації на додаток, що реалізують автоматизацію пошуку і видачі на екран необхідних помилкових даних.

У даному документі представлений текст програми. Програма написана на мові Python. Об'єм пам'яті, що займають програми комплексу, складає 73 Мб. Конфігурація комп'ютера стандартна. Комплекс функціонує в середовищі MS WINDOWS 10/11.

ЗМІСТ

1. СКЛАД ПРОГРАМИ

Програма складається з наступних функцій:

а) check_mistakes: пошук орфографічних помилок у тексті, зберігає результат у словник;

б) check_miswords: пошук недоречних слів у тексті, зберігає результат у словник;

в) check_misphrases: пошук недоречних речень, зберігає результат у словник;

г) highlight_mistake: приймає помилку на вхід, підкреслює помилку у тексті;

д) analyze_text: запускає функції аналізу тексту, у другому потоці відображає поточний стан аналізу та продовжує виконання програми;

е) open_file: відкриває діалогове вікно відкриття файлу;

ж) read_file: приймає шлях до файлу на вхід, конвертує DOCX та PDF файли у текст;

з) save_file: відкриває діалогове вікно збереження файлу;

и) show_mistakes: виводить список орфографічних помилок на екран;

к) show_miswords: виводить список недоречних слів на екран;

л) show_phrases: виводить список недоречних речень на екран.

2. КОД ПРОГРАМИ

```
import json
import os
import re
import threading
import tkinter
from tkinter import filedialog
import customtkinter
import docx
import language_tool_python
import nltk
import openai
import spacy
from PIL import Image
import pypdf
from pydantic import BaseModel



client = openai.OpenAI()



class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        #None __init__ initialisation
        self.progress_bar = None
        self.save_button = None
        self.save_icon_tk = None
        self.loading_label = None
        self.tool = language_tool_python.LanguageTool('uk')  # Вказуємо українську мову
        self.mistake_frames = []
```

```
self.no_mistakes_label = None
self.add_icon_tk = None
self.add_button = None
self.textbox = None
self.analyze_button = None
self.analyze_header_label = None
self.mistakes_button = None
self.miswords_button = None
self.misphrases_button = None
self.mistakes_frame = None
self.filepath = None

self.mistakes_list = []
self.miswords_list = []
self.misphrases_list = []

customtkinter.set_appearance_mode("Dark")
customtkinter.set_default_color_theme("green")

self.geometry("1200x800")
self.title("Аналізатор тексту")
self.grid_columnconfigure(1, weight=1)
self.grid_rowconfigure(0, weight=1)

# Фрейми
self.menu_frame = self.create_menu_frame()
self.textbox_frame = self.create_textbox_frame()
self.analyze_frame = self.create_analyze_frame()

# Грід для фреймів
self.menu_frame.grid(row=0, column=0, padx=(10,0), pady=10, sticky="nsw")
self.textbox_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nswe")
self.textbox_frame.grid_columnconfigure(0, weight=1)
self.textbox_frame.grid_rowconfigure(0, weight=1)
self.analyze_frame.grid_columnconfigure(0, minsize=320)
```

```
        self.analyze_frame.grid_rowconfigure(4, weight=1)
        self.analyze_frame.grid(row=0, column=2, padx=(0,10), pady=10, sticky="nse")

    def create_menu_frame(self):
        menu_frame = customtkinter.CTkFrame(self, width=70)

        ### Елементи інтерфейсу для menu_frame:

        # Кнопка "Додати"
        try:
            self.add_icon_tk = customtkinter.CTkImage(
                light_image=Image.open("icons/add-icon-light.png"),
                dark_image=Image.open("icons/add-icon-dark.png"),
                size=(30, 30))
        except FileNotFoundError:
            print("Помилка: Файл іконки не знайдено.")
            self.add_icon_tk = None

        if self.add_icon_tk:
            self.add_button = customtkinter.CTkButton(
                menu_frame,
                image=self.add_icon_tk,
                text="", compound="left",
                command=self.open_file,
                height=50,
                width=50)
        else:
            self.add_button = customtkinter.CTkButton(
                menu_frame,
                text="+",
                command=self.open_file,
                height=50,
                width=50)
        self.add_button.grid(row=0, column=0, padx=10, pady=(10, 5), sticky="ew")
```

```
    # Кнопка "Зберегти"
    try:
        self.save_icon_tk = customtkinter.CTkImage(
            light_image=Image.open("icons/save-icon-light.png"),
            dark_image=Image.open("icons/save-icon-dark.png"),
            size=(30, 30))
    except FileNotFoundError:
        print("Помилка: Файл іконки не знайдено.")
        self.save_icon_tk = None

    if self.save_icon_tk:
        self.save_button = customtkinter.CTkButton(
            menu_frame,
            image=self.save_icon_tk,
            text="", compound="left",
            command=self.save_file,
            height=50,
            width=50)
    else:
        self.save_button = customtkinter.CTkButton(
            menu_frame,
            text="💾",
            command=self.save_file,
            height=50,
            width=50)
    self.save_button.grid(row=1, column=0, padx=10, pady=(5, 10), sticky="ew")

    # Повернути створений фрейм
    return menu_frame

def create_textbox_frame(self):
    textbox_frame = customtkinter.CTkFrame(self)

    ###  інтерфейсу для textbox_frame:
```

```
        # Текстове поле
        self.textbox = customtkinter.CTkTextbox(
            textbox_frame,
            wrap=tkinter.WORD,
            font=customtkinter.CTkFont(
                family="Times New Roman",
                size=20))
        self.textbox.grid(row=0, column=0, padx=10, pady=(10, 0), sticky="nswe")

        # Скидання модифікаційного флагу (щоб працювала подія <<Modified>>)
        self.textbox.edit_modified(False)

        # Кнопка аналізу
        self.analyze_button = customtkinter.CTkButton(
            textbox_frame,
            border_spacing=8,
            text="Аналізувати",
            command=self.analyze_text,
            font=customtkinter.CTkFont(
                family="Arial",
                size=20,
                weight="bold"
            ))
        self.analyze_button.grid(row=1, column=0, padx=10, pady=10)

        # Повернути створений фрейм
        return textbox_frame

    def create_analyze_frame(self):
        analyze_frame = customtkinter.CTkFrame(self, width=320)

        ### Елементи інтерфейсу для analyze_frame:

        # Заголовок
        self.analyze_header_label = customtkinter.CTkLabel(
```

```
        analyze_frame,
        padx=10, pady=10,
        text="Аналіз тексту:",
        font=customtkinter.CTkFont(
            family="Arial",
            size=20,
            weight="bold"))
    self.analyze_header_label.grid(row=0, column=0, padx=10, pady=10)

    # Кнопки результату аналізу
    self.mistakes_button = customtkinter.CTkButton(
        analyze_frame,
        fg_color="#603a40",
        text_color="#191716",
        border_spacing=8,
        text="Помилки",
        command=self.show_mistakes,
        font=customtkinter.CTkFont(
            family="Arial",
            size=20,
            weight="bold"))
    self.mistakes_button.grid(row=1, column=0, padx=10, pady=(0, 10))

    self.miswords_button = customtkinter.CTkButton(
        analyze_frame,
        fg_color="#84596b",
        text_color="#191716",
        border_spacing=8,
        text="Недоречні слова",
        command=self.show_miswords,
        font=customtkinter.CTkFont(
            family="Arial",
            size=20,
            weight="bold"))
    self.miswords_button.grid(row=2, column=0, padx=10, pady=(0, 10))
```

```
        self.misphrases_button = customtkinter.CTkButton(
            analyze_frame,
            fg_color="#af9bb6",
            text_color="#191716",
            border_spacing=8,
            text="AI аналіз",
            command=self.show_misphrases,
            font=customtkinter.CTkFont(
                family="Arial",
                size=20,
                weight="bold"))
        self.misphrases_button.grid(row=3, column=0, padx=10, pady=(0, 10))

        # Прокручуваний фрейм
        self.mistakes_frame = customtkinter.CTkScrollableFrame(
            analyze_frame,
            width=300,
        )
        analyze_frame.grid_rowconfigure(4, weight=1)
        self.mistakes_frame.grid(row=4, column=0, padx=10, pady=(0, 10), sticky="ns")

        # Повернути створений фрейм
        return analyze_frame

    def save_file(self):
        """Зберігає текст з textbox у файл."""
        filetypes = (("Текстові файли", "*.txt"), ("DOCX файли", "*.docx"))
        filepath = tkinter.filedialog.asksaveasfilename(
            title="Зберегти файл", defaultextension=".txt", filetypes=filetypes)

        if filepath:
            try:
                text = self.textbox.get("1.0", "end-1c")  # Отримуємо текст з textbox
                filename, file_extension = os.path.splitext(filepath)
```

```
                if file_extension == ".txt":
                    with open(filepath, "w", encoding="utf-8") as file:
                        file.write(text)
                elif file_extension == ".docx":
                    doc = docx.Document()
                    for line in text.splitlines():
                        doc.add_paragraph(line)
                    doc.save(filepath)
                else:
                    print("Формат файлу не підтримується.")
            except Exception as e:
                print(f"Помилка збереження файлу: {e}")


    def open_file(self):
        filetypes = (("Текстові файли", "*.txt"), ("PDF файли", "*.pdf"), ("DOCX файли", "*.docx"), ("Усі файли", "*.*"))
        filepath = tkinter.filedialog.askopenfilename(title="Вибрати файл", filetypes=filetypes)

        if filepath:
            self.filepath = filepath
            try:
                text = self.read_file(filepath)
                self.textbox.delete("1.0", "end")
                self.textbox.insert("1.0", text)
                #print(f"Вміст файлу {filepath}:\n{text}")
            except Exception as e:
                print(f"Помилка відкриття файлу: {e}")
                self.textbox.delete("1.0", "end")
                self.textbox.insert("1.0", f"Помилка відкриття файлу: {e}")


    @staticmethod
    def read_file(filepath):
        filename, file_extension = os.path.splitext(filepath)
        if file_extension == ".txt":
```

```
            with open(filepath, "r", encoding="utf-8") as file:
                return file.read()
        elif file_extension == ".pdf":
            reader = pypdf.PdfReader(filepath)
            text = ""
            for page in reader.pages:
                text += page.extract_text()
            return text
        elif file_extension == ".docx":
            doc = docx.Document(filepath)
            text = ""
            for paragraph in doc.paragraphs:
                text += paragraph.text + "\n"
            return text
        else:
            return "Формат файлу не підтримується."

    def analyze_text(self):
        def long_task():
            # Выполняем анализ текста один раз
            text = self.textbox.get("1.0", "end")
            self.check_mistakes(text)
            self.progress_bar.set(0.33)
            self.check_miswords(text)
            self.progress_bar.set(0.66)
            self.check_misphrases(text)
            self.progress_bar.set(1.0)

            self.progress_bar.destroy()

            # Включаем кнопку обратно
            self.analyze_button.configure(state="normal")

        # Создаем и отображаем прогресс-бар
        self.progress_bar = customtkinter.CTkProgressBar(self)
```

```
        self.progress_bar.grid(row=1, column=0, columnspan=2, pady=10, padx=20,
sticky="ew")

        # Отключаем кнопку, чтобы предотвратить повторный запуск
        self.analyze_button.configure(state="disabled")
        self.progress_bar.set(0)

        # Запускаем длительную задачу в отдельном потоке
        analysis_thread = threading.Thread(target=long_task)
        analysis_thread.start()

    def check_mistakes(self, text):
        """Перевіряє текст на орфографічні помилки за допомогою LanguageTool"""
        self.mistakes_list.clear()

        matches = self.tool.check(text)  # Виконуємо перевірку тексту

        filtered_matches = [match for match in matches
                            if match.ruleId != "UK_MIXED_ALPHABETS"
                            and match.ruleId != "WHITESPACE_RULE"
                            and match.ruleId != "COMMA_PARENTHESIS_WHITESPACE"
                            and match.ruleId != "UK_SIMPLE_REPLACE_SPELLING_1992"]

        for match in filtered_matches:
            print(json.dumps(vars(match), indent=4, ensure_ascii=False))
            print("-" * 20)  # Роздільник між об'єктами

        for match in filtered_matches:
            self.mistakes_list.append({
                "message": match.message,  # Опис проблеми
                "replacements": match.replacements,  # Варіанти виправлення
                "offset": match.offset,  # Позиція у тексті
                "errorLength": match.errorLength,  # Довжина помилки
            })

        num_mistakes = len(self.mistakes_list)
```

```
        self.mistakes_button.configure(text=f"Помилки ({num_mistakes})")
        print("Перевірка помилок")
        #print(text)

    def show_mistakes(self):
        #
        if len(self.mistakes_frame.winfo_children()) != 0:
            for widget in self.mistakes_frame.winfo_children():
                widget.destroy()
            self.textbox.tag_remove("highlight", "1.0", "end")
            return

        if not self.mistakes_list:
            # Якщо список порожній, виводимо повідомлення
            self.no_mistakes_label = customtkinter.CTkLabel(
                self.mistakes_frame,
                text="Помилок не знайдено!",
                font=customtkinter.CTkFont(
                    family="Arial",
                    size=16),
            )
            self.no_mistakes_label.grid(row=0, column=0, padx=10, pady=5, sticky="n")
            return

        # Додати підфрейми для кожної помилки
        for idx, mistake in enumerate(self.mistakes_list):
            mistake_button = CustomMistakesButton(
                self.mistakes_frame,
                text1=f"Помилка {idx + 1}: {mistake['message']}",
                text2=f"Пропозиції: {', '.join(mistake['replacements']) if
mistake['replacements'] else 'Немає'}",
                command=lambda m=mistake: self.highlight_mistake(m))

            mistake_button.grid(row=idx, column=0, padx=(10,5), pady=(5,5),
sticky="ew")
            mistake_button.grid_columnconfigure(0, weight=1)
```

```
            self.mistake_frames.append(mistake_button)

    def highlight_mistake(self, mistake):
        """Виділяє текст у текстбоксі на основі помилки."""
        offset = mistake['offset']
        error_length = mistake['errorLength']

        # Видалити старе виділення
        self.textbox.tag_remove("highlight", "1.0", "end")

        # Обчислити позиції для виділення
        start = f"1.0 + {offset} chars"
        end = f"{start} + {error_length} chars"

        # Додати тег для виділення
        self.textbox.tag_add("highlight", start, end)

        # Налаштувати колір тега
        self.textbox.tag_config("highlight", background="#af9bb6", foreground="white")

        # Прокрутити текстбокс до виділеного тексту
        self.textbox.see(start)

    def check_miswords(self, text):
        self.miswords_list.clear()

        vague_words = {
            "Нечіткі формулювання": [
                "можливий", "ймовірний", "деякий", "багато", "необхідний",
"приблизний",
                "достатній", "частий", "як правило", "в значній мірі", "практично",
                "майже", "орієнтовний", "немалий", "більший", "зазвичай", "по
можливості",
                "в цілому", "більшість", "кілька", "чимало", "певний", "деякі",
"іноді",
                "нерідко", "в окремих випадках", "в деяких випадках", "залежно від",
```

```
                "в залежності від", "відповідно до", "згідно з", "у відповідності до",
                "на основі", "базуючись на", "виходячи з", "включаючи", "зокрема",
                "наприклад", "та інше", "тощо", "та інші", "і т.д.", "і т.п.",
"близько",
                "порядку", "коло", "приблизно", "дещо", "частково", "майже завжди",
                "досить часто", "рідко", "дуже рідко", "вкрай рідко", "часом",
                "періодично", "регулярно", "систематично", "хаотично", "спорадично",
                "змінно", "нестабільно", "небайдужий", "особливий"
            ],
            "Суб'єктивні оцінки": [
                "зручний", "простий", "складний", "ефективний", "сучасний", "якісний",
                "важливий", "необхідний", "кращий", "гірший", "оптимальний",
"максимальний",
                "мінімальний", "доцільний", "актуальний", "корисний", "зрозумілий",
                "безпечний", "надійний", "стабільний", "приємний",
                "гарний", "поганий", "цікавий", "нудний", "швидкий", "повільний",
                "легкий", "важкий", "великий", "малий", "високий", "низький",
"значний",
                "незначний", "достатній", "недостатній", "комфортний", "некомфортний",
                "зручно", "незручно", "просто", "складно", "ефективно", "неефективно",
                "сучасно", "застаріло", "якісно", "неякісно"
            ],
            "Слова, що вимагають уточнення": [
                "швидкий", "повільний", "легкий", "важкий", "мало", "нещодавній",
                "скоро", "в майбутньому", "зараз", "сьогодні", "завтра", "пізніше",
                "раніше", "тривалий", "короткий", "довгий", "близький", "далекий",
                "великий", "малий", "багато", "кілька", "деякий час", "деякий період",
                "деяка кількість", "деяка частина", "різноманітний", "нужденний",
"популярний"
            ]
        }

        try:
            nlp = spacy.load("uk_core_news_sm")
        except OSError:
            print("Будь ласка, завантажте модель uk_core_news_sm через spacy-cli.")
            exit()
```

```
        doc = nlp(text)

        for category, words in vague_words.items():
            for token in doc:
                # Перевірка слова в початковій формі та у всіх формах на відповідність "проблемним" словам
                test = token.lemma_
                if token.lemma_ in words or token.text.lower() in words:
                    self.miswords_list.append({
                        'word': token.text,
                        'offset': token.idx,
                        'errorLength': len(token.text),
                        'category': category
                    })

        # Вивід результатів
        print(json.dumps(self.miswords_list, indent=4, ensure_ascii=False))

        num_miswords = len(self.miswords_list)
        self.miswords_button.configure(text=f"Недоречні слова ({num_miswords})")

    def show_miswords(self):
        #
        if len(self.mistakes_frame.winfo_children()) != 0:
            for widget in self.mistakes_frame.winfo_children():
                widget.destroy()
            self.textbox.tag_remove("highlight", "1.0", "end")
            return

        if not self.miswords_list:
            # Якщо список порожній, виводимо повідомлення
            self.no_mistakes_label = customtkinter.CTkLabel(
                self.mistakes_frame,
                text="Недоречних слів не знайдено!",
                font=customtkinter.CTkFont(
```

```
                family="Arial",
                size=16),
            )
            self.no_mistakes_label.grid(row=0, column=0, padx=10, pady=5, sticky="n")
            return

        # Додати підфрейми для кожної помилки
        for idx, misword in enumerate(self.miswords_list):
            mistake_button = CustomMistakesButton(
                self.mistakes_frame,
                text1=f"Недоречне слово {idx + 1}: {misword['word']}",
                text2=f"Категорія: {misword['category']}",
                command=lambda m=misword: self.highlight_mistake(m))

            mistake_button.grid(row=idx, column=0, padx=(10,5), pady=(5,5),
sticky="ew")
            mistake_button.grid_columnconfigure(0, weight=1)
            mistake_button.grid_rowconfigure(idx, weight=1)
            self.mistake_frames.append(mistake_button)

    def check_misphrases(self, text):
        """
        Аналізує речення за допомогою OpenAI API на доречність та змістовність.
        """
        self.misphrases_list.clear()

        text = text.strip()
        if not text:
            print("Немає тексту для аналізу.")
            return

        class SentenceAnalysis(BaseModel):
            is_relevant: bool
            comment: str

        try:
```

```
            # Розбиваємо текст на речення
            sentences = nltk.sent_tokenize(text)

            # Зберігаємо offset і sentenceLength для кожного речення
            sentence_data = []
            current_offset = 0

            for one_of_sentences in sentences:
                if re.match(r"^\d+(\.\d+)*\.$", one_of_sentences):
                    current_offset += len(one_of_sentences) + 1
                    continue  # Ігноруємо це речення
                sentence_length = len(one_of_sentences)
                sentence_data.append({
                    'sentence': one_of_sentences,
                    'offset': current_offset,
                    'errorLength': sentence_length
                })
                current_offset += sentence_length + 1  # +1 для пробілу чи розділового знака між реченнями

            history = [
                {
                    "role": "system",
                    "content":
                        # "You are an experienced web developer and a specialist in creating technical requirements (TR). "
                        # "Your goal is to analyze the TR text, evaluating each sentence according to the following criteria:\n\n"
                        # "1. **Whether the sentence corresponds to the topic 'Website Requirements'** (yes/no).\n"
                        # "2. **Comment on the content**: if the sentence is irrelevant, provide a concise comment explaining why and suggest an improvement, "
                        # "but limit the comment to no more than two sentences. Do not add any comments for sentences that are relevant."
                        # "Please provide all answers in Ukrainian.\n\n"
                        # "\n\nIn addition, evaluate whether each sentence meets the following standards:\n"
```

```
                        # "- **Specificity**: Sentences should avoid vague or abstract
terms like 'very fast' or 'beautiful'; instead, they should specify measurable criteria
(e.g., 'The website should load within 2 seconds').\n"

                        # "- **Actionability**: Each requirement should clearly
describe an actionable and achievable goal for the development team.\n"

                        # "- **Relevance**: Sentences should directly contribute to the
topic 'Website Requirements'.\n\n"

                        # "Examples of sentences that are NOT acceptable:\n"

                        # "- 'The website should be very fast and beautiful.' (Too
vague; lacks measurable criteria.)\n"

                        # "- 'We want a perfect website.' (Abstract and
unachievable.)\n\n"

                        # "Please analyze each sentence individually, clearly, and in a
structured manner.",

                    """ You are an experienced web developer and a specialist in
creating technical requirements (TR).

                        Your goal is to analyze the TR text, evaluating each sentence
according to the following criteria:



                        1. **Whether the sentence corresponds to the topic 'Website
Requirements'** (yes/no).

                        2. **Comment on the content**: if the sentence is irrelevant,
provide a concise comment explaining why and suggest an improvement, but limit the
comment to no more than two sentences.

                        Do not add any comments for sentences that are relevant.

                        Please provide all answers in Ukrainian.



                        **Important Notes**:

                        – **Headings or section titles** should always be considered
relevant. For example:

                        – "Functional Requirements"

                        – "Performance Standards"

                        These phrases indicate sections or headings, and no further
analysis of their content is needed.

                        – Evaluate other sentences based on the following standards:

                        – **Specificity**: Sentences should avoid vague or abstract
terms like 'very fast' or 'beautiful'; instead, they should specify measurable criteria
(e.g., 'The website should load within 2 seconds').

                        – **Actionability**: Each requirement should clearly describe
an actionable and achievable goal for the development team.

                        – **Relevance**: Sentences should directly contribute to the
topic 'Website Requirements'.
```

```
                        Examples of sentences that are NOT acceptable:

                        – 'The website should be very fast and beautiful.' (Too vague; lacks measurable criteria.)

                        – 'We want a perfect website.' (Abstract and unachievable.)


                        Please analyze each sentence individually, clearly, and in a structured manner, ensuring that section titles or headings are automatically marked as relevant without requiring further content analysis."""

                }

            ]


            for sentence_info in sentence_data:

                # Відправка запиту до GPT для кожного речення

                # response = client.beta.chat.completions.parse(

                #     model="gpt-4o-mini",

                #     messages=[

                #         {

                #             "role": "system",

                #             "content":

                #                 "You are an experienced web developer and a specialist in creating technical requirements (TR). "

                #                 "Your goal is to analyze the TR text, evaluating each sentence according to the following criteria:\n\n"

                #                 "1. **Whether the sentence corresponds to the topic 'Website Requirements'** (yes/no).\n"

                #                 "2. **Comment on the content**: if the sentence is irrelevant, provide a concise comment explaining why and suggest an improvement, "

                #                 "but limit the comment to no more than two sentences. Do not add any comments for sentences that are relevant."

                #                 "Please provide all answers in Ukrainian.\n\n"

                #                 "\n\nIn addition, evaluate whether each sentence meets the following standards:\n"

                #                 "- **Specificity**: Sentences should avoid vague or abstract terms like 'very fast' or 'beautiful'; instead, they should specify measurable criteria (e.g., 'The website should load within 2 seconds').\n"

                #                 "- **Actionability**: Each requirement should clearly describe an actionable and achievable goal for the development team.\n"

                #                 "- **Relevance**: Sentences should directly contribute to the topic 'Website Requirements'.\n\n"
```

```
                #                 "Examples of sentences that are NOT acceptable:\n"
                #                 "- 'The website should be very fast and beautiful.' (Too vague; lacks measurable criteria.)\n"
                #                 "- 'We want a perfect website.' (Abstract and unachievable.)\n\n"
                #                 "Please analyze each sentence individually, clearly, and in a structured manner."
                #
                #         },
                #         {
                #             "role": "user",
                #             "content": f"Речення з тексту вимог до сайту: \n\n{sentence_info['sentence']}\n\n Як би ти оцінив це речення?"
                #         }
                #     ],
                #     response_format=SentenceAnalysis
                # )

                history.append({
                    "role": "user",
                    #"content": f"Речення з тексту вимог до сайту: \n\n{sentence_info['sentence']}\n\n Як би ти оцінив це речення?"
                    "content": f"Текст вимог до сайту:\n\n{text}\n\nЯк би ти оцінив це речення?\n{sentence_info['sentence']}"
                })

                response = client.beta.chat.completions.parse(
                    model="gpt-4o-mini",
                    messages=history,  # Історія повідомлень
                    response_format=SentenceAnalysis
                )

                # Обробка результату
                analysis = response.choices[0].message.parsed
                print(analysis)
                if not analysis.is_relevant:
                    self.misphrases_list.append({
                        "message": analysis.comment,  # Коментар щодо проблеми
```

```
                    "offset": sentence_info['offset'],  # Позиція речення у тексті
                    "errorLength": sentence_info['errorLength'],  # Довжина речення
                })

            num_mistakes = len(self.misphrases_list)
            self.misphrases_button.configure(text=f"AI аналіз ({num_mistakes})")
            print("AI аналіз")

        except Exception as e:
            print(f"Помилка аналізу речень: {e}")
            self.misphrases_button.configure(text="AI аналіз (помилка)")

    def show_misphrases(self):
        #
        if len(self.mistakes_frame.winfo_children()) != 0:
            for widget in self.mistakes_frame.winfo_children():
                widget.destroy()
            self.textbox.tag_remove("highlight", "1.0", "end")
            return

        if not self.misphrases_list:
            # Якщо список порожній, виводимо повідомлення
            self.no_mistakes_label = customtkinter.CTkLabel(
                self.mistakes_frame,
                text="Недоречних речень не знайдено!",
                font=customtkinter.CTkFont(
                    family="Arial",
                    size=16),
            )
            self.no_mistakes_label.grid(row=0, column=0, padx=10, pady=5, sticky="n")
            return

        # Додати підфрейми для кожної помилки
        for idx, misphrase in enumerate(self.misphrases_list):
            mistake_button = CustomMistakesButton(
```

```
                self.mistakes_frame,
                text1=f"Є проблема в речені {idx + 1}:",
                text2=f"{misphrase['message']}",
                command=lambda m=misphrase: self.highlight_mistake(m))

            mistake_button.grid(row=idx, column=0, padx=(10,5), pady=(5,5),
sticky="ew")
            mistake_button.grid_columnconfigure(0, weight=1)
            mistake_button.grid_rowconfigure(idx, weight=1)
            self.mistake_frames.append(mistake_button)

class CustomMistakesButton(customtkinter.CTkFrame):
    def __init__(self, parent, text1, text2, command=None, **kwargs):
        super().__init__(parent, **kwargs)

        self._corner_radius = 5
        self.defaultColor = self._fg_color

        # Основний фрейм (фон кнопки)
        self.command = command
        self.configure(cursor="hand2")  # Змінюємо курсор на "руку" при наведенні

        # Верхній текст
        self.mistake_message_label = customtkinter.CTkLabel(
            self,
            text=text1,
            font=customtkinter.CTkFont(
                size=14,
                family="Arial",
                weight="bold"),
            state="disabled",
            wraplength=270,
        )
        self.mistake_message_label.grid(row=0, column=0, padx=5, pady=(5, 0),
sticky="w")
```

```
        # Нижній текст
        self.mistake_suggestion_label = customtkinter.CTkLabel(
            self,
            text=text2,
            font=customtkinter.CTkFont(
                size=14,
                family="Arial"),
            wraplength=270,
        )
        self.mistake_suggestion_label.grid(row=1, column=0, padx=5, pady=5, sticky="w")

        # Прив'язка кліків
        self.bind("<Button-1>", self.on_click)
        self.mistake_message_label.bind("<Button-1>", self.on_click)
        self.mistake_suggestion_label.bind("<Button-1>", self.on_click)

        # Прив'язка подій hover
        self.bind("<Enter>", self.on_hover)
        self.bind("<Leave>", self.on_leave)
        self.mistake_message_label.bind("<Enter>", self.on_hover)
        self.mistake_message_label.bind("<Leave>", self.on_leave)
        self.mistake_suggestion_label.bind("<Enter>", self.on_hover)
        self.mistake_suggestion_label.bind("<Leave>", self.on_leave)

    def on_click(self, event=None):
        if self.command:
            self.command()

    def on_hover(self, event=None):
        self.configure(fg_color="gray")

    def on_leave(self, event=None):
        self.configure(fg_color=self.defaultColor)

if __name__ == "__main__":
```

```
app = App()
app.mainloop()
```

ДОДАТОК В

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАТВЕРДЖУЮ
Проректор Українського
державного університету
науки і технологій
Анатолій РАДКЕВИЧ
28.12.24

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Керівництво користувача. Керівництво з аналізу вимог
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.01444-01 13 01-ЛЗ

Представники
підприємства-розробника
Завідувач кафедри КІТ
Вадим ГОРЯЧКІН
28.12.24

Керівник розробки
Олена КУРОП’ЯТНИК
28.12.24

Виконавець

Нікіта ЛОХВИЦЬКИЙ

28.12.24

Нормоконтролер
Світлана ВОЛКОВА
28.12.24

2025

ЗАТВЕРДЖЕНО
44165850.01444-01 13 01-ЛЗ

АВТОМАТИЗОВАНИЙ АНАЛІЗАТОР ТЕКСТУ ВИМОГ ДО САЙТІВ

Керівництво користувача. Керівництво з аналізу вимог
44165850.01444-01 13 01

Листів 14

2025

АНОТАЦІЯ

Документ 44165850.01444-01 13 01 «Автоматизований аналізатор тексту вимог до сайтів. Керівництво користувача. Керівництво з аналізу вимог» входить до складу програмної документації на програму для автоматизації пошуку і видачі на екран необхідних помилкових даних.

У даному документі представлено керівництво користувача. Програма написана на мові Python. Об'єм пам'яті, що займає програма, складає 73 мб. місця на накопичувачі. Конфігурація комп'ютера стандартна. Комплекс функціонує під управлінням ОС Windows 10\11.

ЗМІСТ

ВСТУП

Цей посібник призначений для ознайомлення користувачів з програмою "Автоматизований аналізатор тексту вимог до сайтів". Він містить детальну інформацію про функціональність програми, її інтерфейс та основні принципи роботи. Метою цього посібника є надання користувачам необхідних знань та навичок для ефективного використання програми " Автоматизований аналізатор тексту вимог до сайтів " та досягнення бажаних результатів.

1. ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

Програма призначена для автоматизованої перевірки якості текстових вимог до веб-сайтів. Вона дозволяє виявляти потенційні помилки, неточності, неоднозначності та інші проблеми у формулюваннях вимог на ранніх етапах розробки, що сприяє підвищенню якості кінцевого продукту та зменшенню витрат на його створення. Програма може бути використана аналітиками, розробниками, тестувальниками та іншими фахівцями, залученими до процесу розробки веб-сайтів.

Вимоги до функціональних характеристик наступні:

а) завантаження текстових файлів у форматах TXT, DOCX і PDF;

б) аналізувати текст за кількома параметрами: орфографія; неоднозначні, суб’єктивні та неточні слова; неоднозначні, суб’єктивні та неточні речення. Формулювання є нечітким, якщо його не можливо точно виміряти. Формулювання є неоднозначним, якщо воно має більше одного трактувань. Формулювання є суб’єктивним, якщо його оцінка залежить від суб’єкту, що оцінює;

в) відображати результати аналізу у вигляді звіту з зазначенням типу помилок, їх розташування та можливих виправлень.

Вимоги до надійності наступні:

а) наявність архівної копії тексту програми на зовнішньому носії;

б) забезпечення стійкого функціонування програми;

в) контроль вхідної і вихідної інформації;

г) захист від копіювання ПЗ.

Вимоги до кліматичних умов: температура – 21-25 °С, відносна вологість 40-60%.

Обслуговування не потрібне.

Для роботи із ПЗ достатньо однієї людини, що має досвід роботи із ПК та ознайомлена із керівництвом користувача.

Склад технічних засобів:

а) процесор з тактовою частотою 1 ГГЦ або вище;

б) доступ до мережі Інтернет;

в) 73 мб. місця на накопичувачі;

г) 2000 мб. оперативної пам’яті.

Програма має функціонувати під управлінням ОС Windows 10\11.

Програма реалізована мовою програмування Python, яка має декілька переваг. Однією з них є наявність мовних бібліотек обробки тексту, що є необхідною умовою для вирішення технічного завдання.

2. ПІДГОТОВКА ДО РОБОТИ

Щоб запустити додаток необхідно мати на комп'ютері файл «main» формату «.exe»

3. ОПИС ОПЕРАЦІЙ

Для початку роботи запустіть програму. Запуск програми виконується за допомого подвійного натискання лівою кнопкою миші на виконуваному файлі .exe. Після цього ви потрапите у головне меню програми, яке наведено на рисунку 3.1.

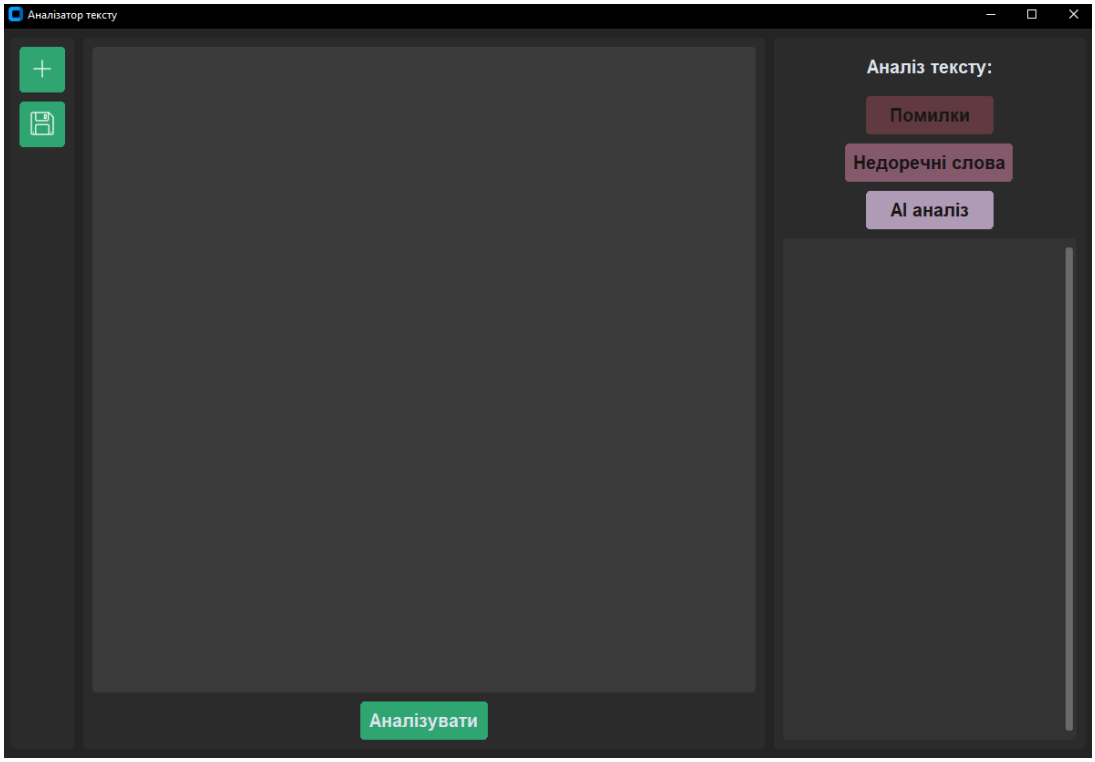

Рисунок 3.1 – головне меню програми

Наразі ви не бачите вимог до сайту та знайдених помилок. Щоб почати роботу у застосунку створіть текст у текстовому полі посередині, або імпортуйте вже існуючий текст. Імпортуйте текст за допомогою кнопки «+» у верхньому лівому куті. Після натискання на кнопку ви потрапите у діалогове вікно, яке наведено на рисунку 3.2.

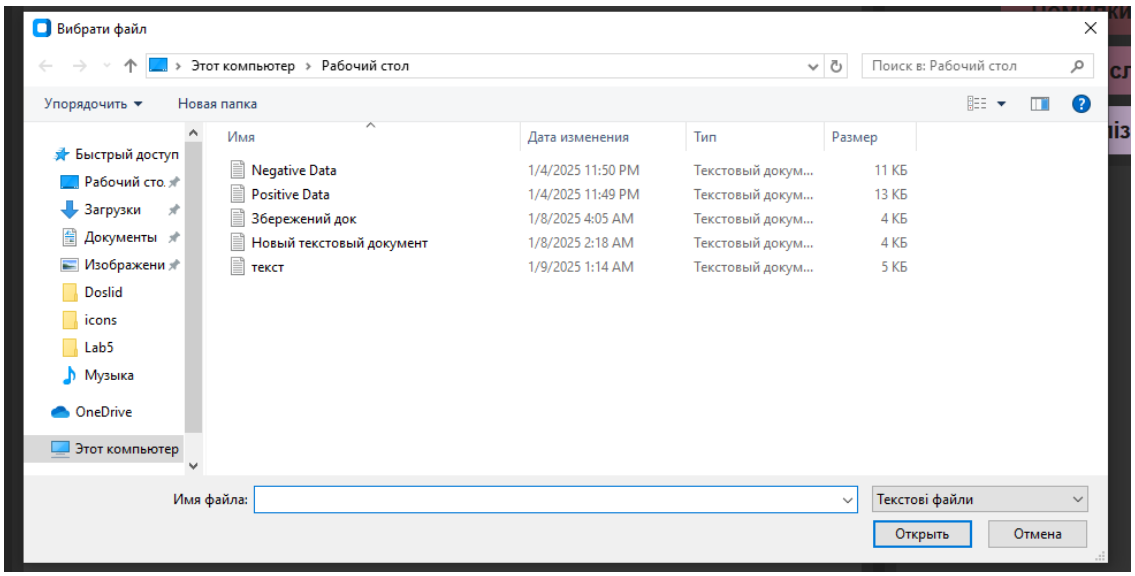

Рисунок 3.2 – Діалогове вікно зчитування вимог

Ви можете обирати між типами обираємого файлу, а саме txt, docx, pdf. Оберіть файл, після чого отримайте результат у вигляді заповненого текстового поля. Текст можна додаткового відкоригувати у текстовому полі. Після завершення всіх змін натисніть кнопку «Аналізувати». Після певного проміжку часу зафіксуйте зміни – на кнопках «Помилки», «Недоречні слова» та «AI аналіз» з’являться числа, що вказують на кількість знайдених недоліків. Дані зміни наведено у рисунку 3.3.

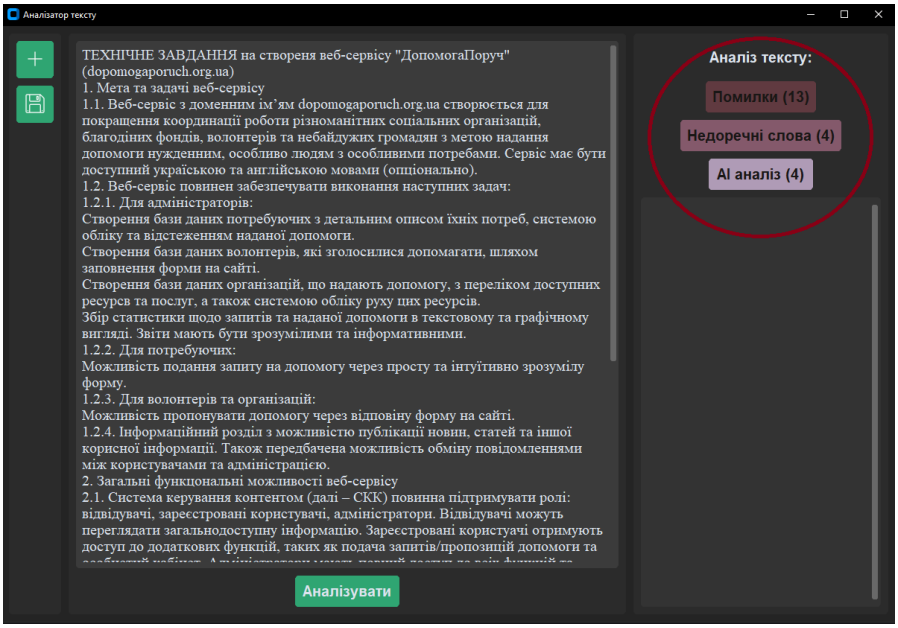

Рисунок 3.3 – Результат аналізу вимог до сайту

Щоб подивитись конкретні недоліки, натисніть кнопку зацікавленого недоліку. При натисканні кнопки «Помилки (13)» ви побачите список помилок,

які слід усунути у вимогах до сайту. Натискання на кожний недолік посилає на конкретне слово у текстовому полі и виділяє це слово кольором.

Аналогічна робота прослідковується з двома іншими кнопками. При натисканні кнопки «Недоречні слова (4)» ви побачите список недоречних слів, а при натисканні «AI аналіз (4)» — список недоречних речень, що зображено на рисунку 3.4.

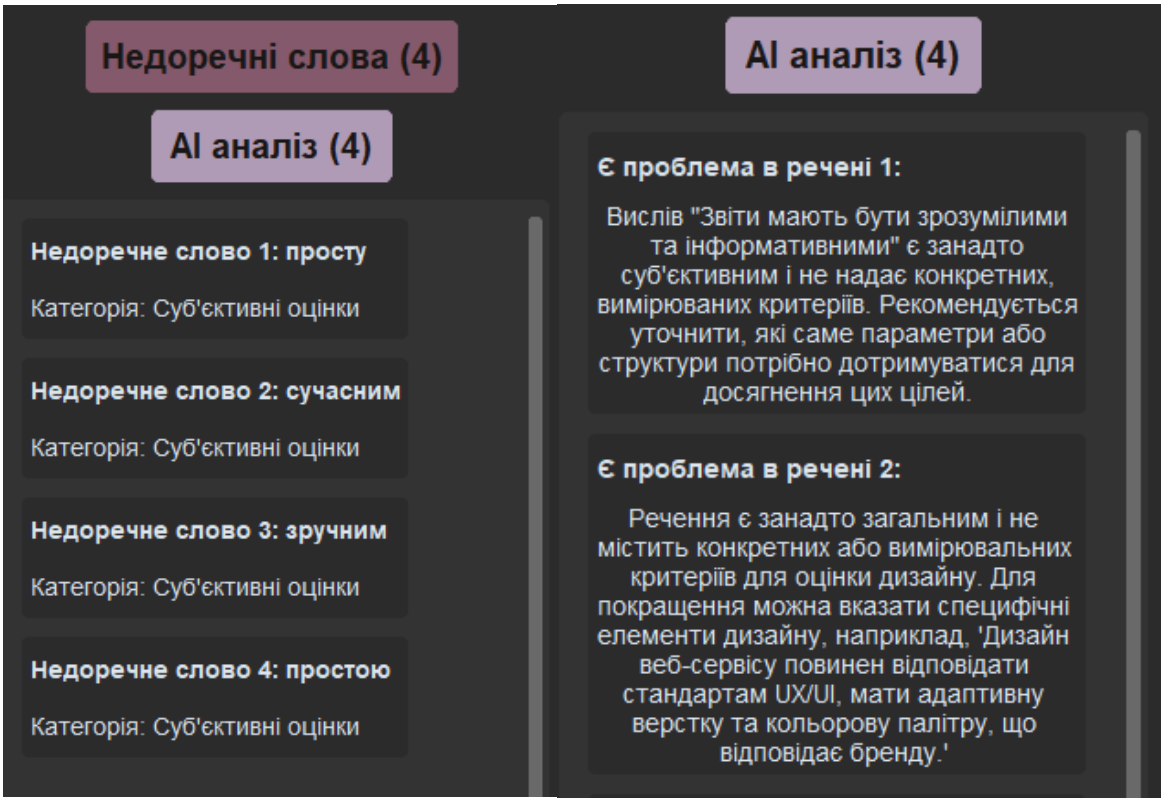

Рисунок 3.4 – Списки недоречних слів та речень

Після того, як побачите недоліки, усуньте у текстовому полі. Щоб зберегти зміни натисніть кнопку «Зберегти», яку розташовано під кнопкою «Додати». Ви потрапите у діалогове вікно, де ви можете зберегти свої зміни, що проілюстровано у рисунку 3.5.

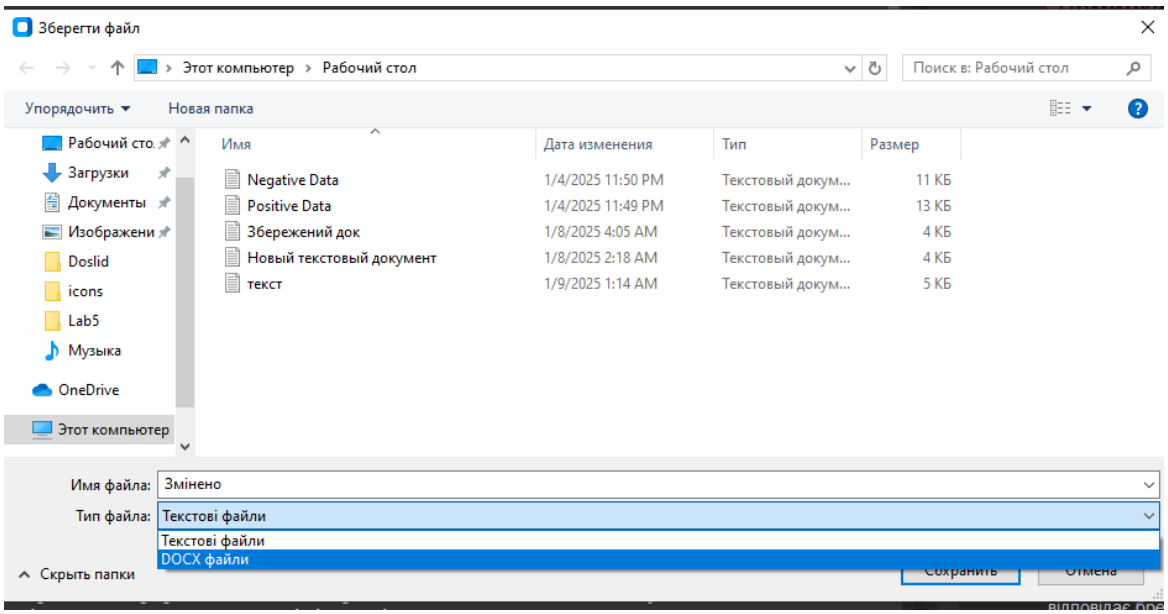

Рисунок 3.5 – Діалогове вікно збереження вимог до сайту

Збережіть дані у текстовому форматі, або у форматі docx для подальшого користування.

4. АВАРІЙНІ СИТУАЦІЇ

Програмний засіб виведе повідомлення про помилку, якщо буде запускатися з пристрою, що не відповідає технічних вимогам.

5. РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Для досягнення найкращих результатів рекомендується використовувати програму на етапах збору, аналізу та документування вимог. Аналіз слід проводити для всіх ключових документів, що містять вимоги до веб-сайту. Рекомендується поєднувати автоматичний аналіз з ручними методами верифікації, такими як перегляди та інспекції, для забезпечення всебічної перевірки якості вимог.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. — 38 с.

ДОДАТОК Г

Тестовий набір вимог до сайтів

ТЕХНІЧНЕ ЗАВДАННЯ на створеня веб-сервісу "ДопомогаПоруч" (dopomogaporuch.org.ua)

1. Мета та задачі веб-сервісу

1.1. Веб-сервіс з доменним ім’ям dopomogaporuch.org.ua створюється для покращення координації роботи різноманітних соціальних організацій, благодіних фондів, волонтерів та небайдужих громадян з метою надання допомоги нужденним, особливо людям з особливими потребами. Сервіс має бути доступний українською та англійською мовами (опціонально).

1.2. Веб-сервіс повинен забезпечувати виконання наступних задач:

1.2.1. Для адміністраторів:

Створення бази даних потребуючих з детальним описом їхніх потреб, системою обліку та відстеженням наданої допомоги.

Створення бази даних волонтерів, які зголосилися допомагати, шляхом заповнення форми на сайті.

Створення бази даних організацій, що надають допомогу, з переліком доступних ресурсв та послуг, а також системою обліку руху цих ресурсів.

Збір статистики щодо запитів та наданої допомоги в текстовому та графічному вигляді. Звіти мають бути зрозумілими та інформативними.

1.2.2. Для потребуючих:

Можливість подання запиту на допомогу через просту та інтуїтивно зрозумілу форму.

1.2.3. Для волонтерів та організацій:

Можливість пропонувати допомогу через відповіну форму на сайті.

1.2.4. Інформаційний розділ з можливістю публікації новин, статей та іншої корисної інформації. Також передбачена можливість обміну повідомленнями між користувачами та адміністрацією.

2. Загальні функцональні можливості веб-сервісу

2.1. Система керування контентом (далі – СКК) повинна підтримувати ролі: відвідувачі, зареєстровані користувачі, адміністратори. Відвідувачі можуть переглядати загальнодоступну інформацію. Зареєстровані користуачі отримують доступ до додаткових функцій, таких як подача запитів/пропозицій допомоги та особистий кабінет. Адміністратори мають повний доступ до всіх функцій та керування сайтом.

2.2. Функціональність веб-сервісу базується на "Типах контенту" (далі – ТК):

Потреби

Пропозиції

Організації

Користувачі

Новини

Статті

Контакти та інша інформація

Кожен ТК має набір полів для заповнення та налаштування доступу. ТК "Потреби" пов'язується з ТК "Користувачі", а ТК "Пропозиції" – з ТК "Організації".

2.3. Дизайн веб-сервісу повинен бути сучасним, привабливим та зручним для користувачів.

2.4. Верстка сайту має бути адапивною та коректно відображатися на всіх популярних браузерах та пристроях.

2.5. Сайт повинен швидко завантажуватися та бути оптимізованим для пошукових систем.

2.6. Безпека сайту повинна бути на високому рівні.

2.7. Реєстрація користувачів повина бути простою та зрозумілою.