МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ	економіки та бізнес-освіти
Кафедра	економіки та цифрового бізнесу
Спеціальність	122 «Комп'ютерні науки»
Форма навчання	Заочна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Бази Романа Юрійовича
(прізвище, ім'я, по батькові здобувача)

на тему Розробка веб-ресурсу для формування документації програмного коду
(повна назва теми)

за матеріалами _____

(повна назва бази дослідження)

науковий керівник _____ _____ Шокотько Л.М.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 09.06.2025р. № 12
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг – 2025

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384

Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
Завідувач кафедри _____В.М. Радько

"07" квітня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА ЗДОБУВАЧУ
Базі Роману Юрійовичу

1. Тема роботи Розробка веб-ресурсу для формування документації програмного коду

науковий керівник роботи Шокотько Людмила Миколаївна
затверджені наказом вищого навчального закладу від «04» квітня 2025 р. № 151-ст

2. Строк подання здобувачем роботи 31.05.2025р.

3. Зміст кваліфікаційної роботи бакалавра, об'єкт, предмет та мета дослідження:

Розділ 1 ТЕОРЕТИЧНІ АСПЕКТИ АВТОМАТИЗАЦІЇ ФОРМУВАННЯ ДОКУМЕНТАЦІЇ КОДУ

Розділ 2 ПРОЄКТУВАННЯ ВЕБ-РЕСУРСУ ДЛЯ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ ДОКУМЕНТАЦІЇ ПРОГРАМНОГО КОДУ

Розділ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-РЕСУРСУ

Об'єкт дослідження – процес документування коду в контексті розробки програмного забезпечення.

Предмет дослідження – веб-ресурс, призначений для автоматичної генерації документації на основі аналізу вихідного коду.

Мета кваліфікаційної роботи бакалавра – розробка веб-ресурсу для автоматизації формування документації коду, що дозволить спростити та оптимізувати процес створення якісної технічної документації для програмних проєктів.

4. Дата видачі завдання 04.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 28.04.2025р.	25.04.2025
2	Підготовка розділу 2	до 16.05.2025р.	15.05.2025
3	Підготовка розділу 3	до 30.05.2025р.	29.05.2025
4	Реєстрація завершеної дипломної роботи	до 31.05.2025р.	30.05.2025
5	Отримання відгуку від наукового керівника	03-04.06.2025р.	04.06.2025
6	Отримання зовнішньої рецензії	05-06.06.2025р.	06.06.2025
7	Перевірка кваліфікаційної роботи на плагіат	02-09.06.2025р.	04.06.2025
8	Попередній захист кваліфікаційної роботи на кафедрі	03.06.2025р.	03.06.2025
9	Допуск кафедрою кваліфікаційної роботи до захисту	09.06.2025р.	09.06.2025
10	Підготовка студента до захисту в ЕК	до 17.06.2025р.	17.06.2025

Завдання підготував науковий керівник Шокотько Л.М.

Завдання одержав здобувач База Р.Ю.

(підпис) (прізвище та ініціали)

РЕФЕРАТ

Кваліфікаційна робота бакалавра: 68 сторінок, 10 рисунків, 2 таблиці, 15 використаних джерел.

Мета роботи: розробка веб-ресурсу для автоматизованого формування документації програмного коду, що має на меті підвищення ефективності та якості документації у процесі розробки програмного забезпечення. Актуальність теми зумовлена необхідністю скорочення часу на створення документації та мінімізації людського фактору при її формуванні.

У ході дослідження було проаналізовано сучасні підходи та інструменти для документування коду, визначено їхні переваги й недоліки. Як результат - обґрунтовано доцільність створення власного рішення на основі технологій ASP.NET Core, Roslyn, DinkToPdf і docx.js, що забезпечують стабільну роботу системи, підтримку кирилиці та гнучкість для подальшої адаптації.

Було реалізовано повноцінний веб-додаток із серверною логікою для обробки запитів, клієнтським інтерфейсом та можливістю експорту документації у формати PDF і Word. Система протестована на стабільність, зручність користування (оцінка інтерфейсу за шкалою SUS — 82/100).

Основні функції веб-ресурсу включають: автоматичний аналіз програмного коду, генерацію та редагування документації, а також її перегляд та експорт. Рішення дозволяє зменшити витрати часу на створення документації до п’яти-семи хвилин, що значно швидше, ніж при ручному підході.

Перспективи розвитку проєкту охоплюють підтримку нових мов програмування, інтеграцію з Git, автоматичне оновлення документації в реальному часі, розширення форматів експорту (Markdown, HTML) та оптимізацію для роботи з великими файлами.

Ключові слова: автоматизація документації, веб-ресурс, програмний код, ASP.NET CORE, CSS, ROSLYN, UI/UX

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

API – Application Programming Interface (Інтерфейс програмування додатків)

ASP.NET Core – Фреймворк для серверної частини.

AST – Abstract Syntax Tree (Абстрактне синтаксичне дерево).

C#– C Sharp (Мова програмування).

CSS – Cascading Style Sheets (Таблиці стилів).

DI – Dependency Injection (Впорскування залежностей).

HTML – HyperText Markup Language (Мова розмітки).

HTTP – HyperText Transfer Protocol (Протокол передачі).

HTTPS – Secure HTTP (Безпечний протокол).

IDE – Integrated Development Environment (Середовище розробки).

JSON – JavaScript Object Notation (Формат даних).

JWT – JSON Web Token (Токен автентифікації).

PDF – Portable Document Format (Формат документів).

RAM – Random Access Memory (Оперативна пам’ять).

REST– Representational State Transfer (Стиль API).

SUS – System Usability Scale (Шкала зручності).

UI – User Interface (Інтерфейс користувача).

UTF-8 – 8-bit Unicode Format (Кодування Unicode).

UX – User Experience (Користувацький досвід).

XML – Extensible Markup Language (Мова розмітки).

ВСТУП

Актуальність теми зумовлена стрімким розвитком інформаційних технологій та зростанням складності програмного забезпечення. У сучасних умовах документування коду є важливим етапом життєвого циклу розробки, оскільки воно сприяє кращому розумінню проєкту, полегшує його підтримку та масштабування. Однак ручне створення документації часто є трудомістким і схильним до помилок, що робить автоматизацію цього процесу необхідною для підвищення продуктивності команд розробників та забезпечення якості кінцевого продукту.

Метою даної роботи є розробка веб-ресурсу для автоматизації формування документації коду, що дозволить спростити та оптимізувати процес створення якісної технічної документації для програмних проєктів. Автоматизація цього процесу спрямована на підвищення ефективності роботи розробників, зменшення витрат часу та забезпечення стандартизації документації.

Об’єктом дослідження є процес документування коду в контексті розробки програмного забезпечення.

Предметом дослідження виступає веб-ресурс, призначений для автоматичної генерації документації на основі аналізу вихідного коду.

Завдання роботи включають:

- аналіз сучасних підходів до автоматизації документування коду;
- проектування структури та функціоналу веб-ресурсу;
- реалізацію системи з урахуванням вимог користувачів;
- тестування розробленого рішення для забезпечення його надійності та зручності використання.

В результаті виконання випускної кваліфікаційної роботи створено повноцінний веб-додаток із серверною логікою для обробки запитів, клієнтським інтерфейсом та можливістю експорту документації у формати PDF і Word.

Рішення дозволяє зменшити витрати часу на створення документації до п’яти-семи хвилин, що значно швидше, ніж при ручному підході.

Структура дипломної роботи складається з трьох розділів. У першому розділі розглядаються теоретичні основи та сучасні інструменти для автоматизації документування. Другий розділ присвячений проектуванню веб-ресурсу. У третьому розділі тестування системи та аналіз отриманих результатів і розробці . Завершують роботу висновки та рекомендації щодо використання розробленого веб-ресурсу.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ АВТОМАТИЗАЦІЇ ФОРМУВАННЯ ДОКУМЕНТАЦІЇ КОДУ

1.1 Поняття та види документації коду

Документація коду – це сукупність текстових матеріалів, що описують структуру, логіку, функціональність та особливості використання програмного коду. Вона є невід'ємною частиною розробки програмного забезпечення, оскільки забезпечує розуміння проєкту як для розробників, так і для інших зацікавлених сторін, таких як тестувальники, менеджери чи нові члени команди. Основне призначення документації полягає в полегшенні підтримки, модифікації та масштабування коду, а також у зменшенні залежності від конкретних осіб, які брали участь у його створенні.

Документація коду може бути представлена в різних формах залежно від її цільового призначення та аудиторії. До основних видів документації належать:

- внутрішня документація – коментарі та пояснення, розміщені безпосередньо в коді, які допомагають розробникам зрозуміти його логіку та призначення окремих блоків;
- зовнішня документація – окремі документи (наприклад, файли README, специфікації, посібники користувача), що описують загальну архітектуру, інструкції з встановлення чи використання програмного забезпечення;
- автоматично згенерована документація – результат роботи спеціалізованих інструментів, які аналізують код і формують структуровані описи (наприклад, API-документацію) на основі коментарів і структури програми.

Таким чином, документація коду є ключовим елементом, що підвищує якість програмного продукту та сприяє ефективній співпраці в команді розробників.

Основні види документації коду розрізняють залежно від їхнього формату, місця розташування та цільової аудиторії:

– коментарі в коді – це внутрішня документація, що розміщується безпосередньо у вихідному коді. Такі коментарі пояснюють призначення окремих функцій, змінних чи блоків коду, роблячи його зрозумілішим для розробників, які працюють із ним. Наприклад, це можуть бути короткі анотації до складних алгоритмів або вказівки щодо використання певних параметрів. Один із прикладів показаний на рисунку 1.1;

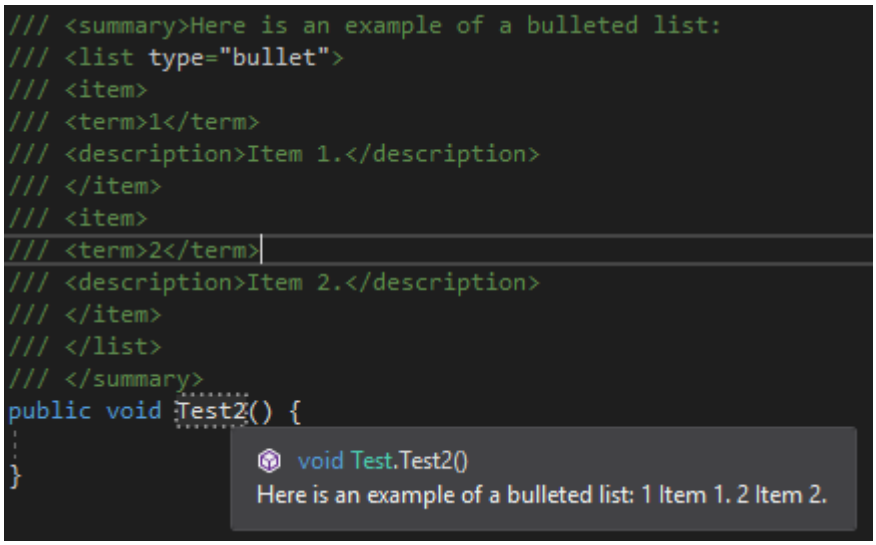

Рис. 1.1. Документація в коді

Джерело: розроблено з використанням [2]

– технічна документація – зовнішні документи, які описують проєкт у цілому. До неї належать описи архітектури системи, інструкції з розгортання, специфікації вимог чи посібники для користувачів. Цей вид документації зазвичай створюється для ширшої аудиторії, включаючи тих, хто не має прямого доступу до коду. Приклад такого документування показаний на рисунку 1.2;

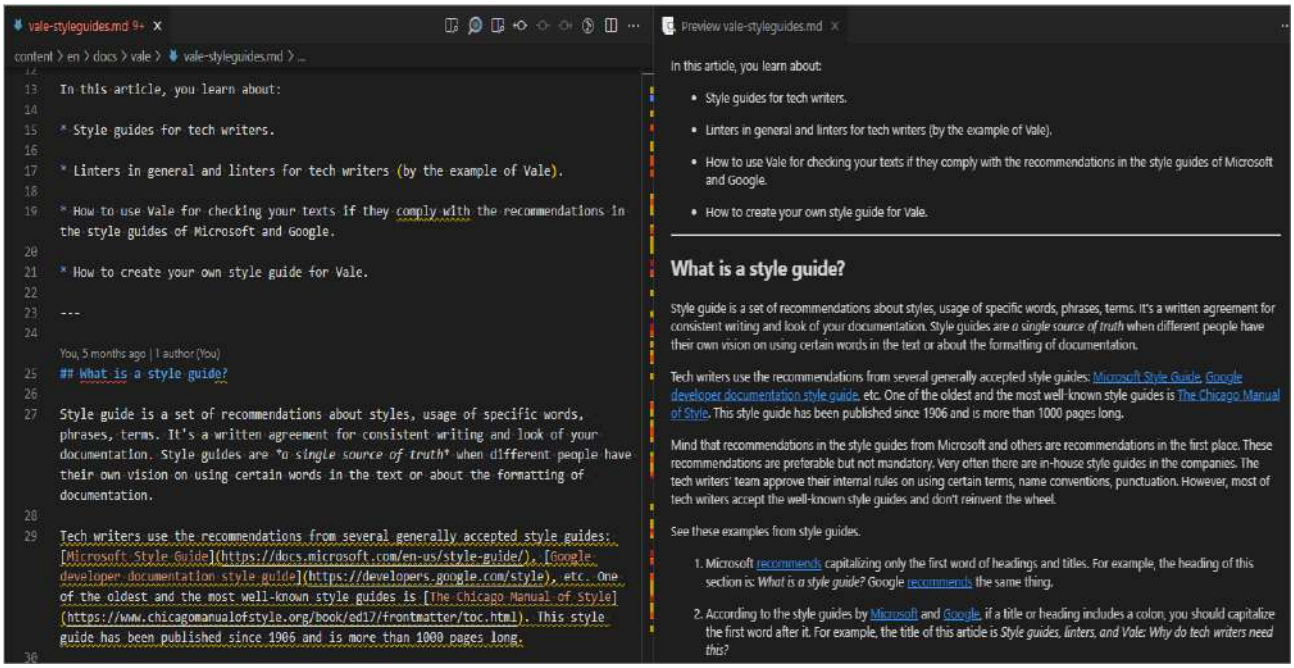

Рис. 1.2. Технічна документація

Джерело: розроблено з використанням [2]

– API-документація – спеціалізований тип документації, що описує інтерфейси програмування додатків (API). Вона включає інформацію про доступні функції, методи, їхні параметри, типи даних і приклади використання. API-документація часто генерується автоматично на основі коментарів у коді й орієнтована на розробників, які інтегрують систему з іншими проєктами. Такий вид документації показаний на рисунку 1.3.

Кожен із цих видів документації відіграє свою роль у процесі розробки, забезпечуючи зручність роботи з кодом і його подальше використання. Вибір конкретного виду залежить від потреб проєкту та його масштабів.

Значення документації для розробки програмного забезпечення.

Документація коду відіграє ключову роль у забезпеченні ефективності та якості процесу розробки програмного забезпечення. По-перше, вона сприяє кращому розумінню коду, що особливо важливо в командній роботі, коли

розробники мають швидко ознайомитися з чужим кодом або повернутися до проєкту через тривалий час.

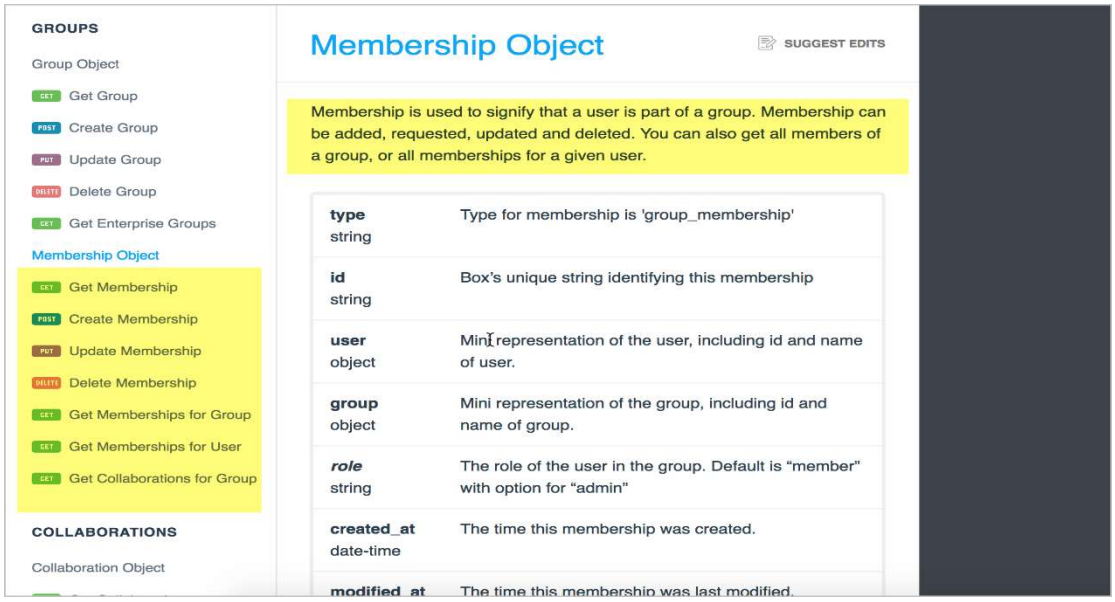

Рис. 1.3. API-документація

Джерело: розроблено з використанням [2]

По-друге, якісна документація зменшує кількість помилок під час модифікації чи інтеграції системи, оскільки розробники отримують чіткі вказівки щодо її структури та функціоналу. По-третє, вона є незамінною для підтримки програмного забезпечення після його релізу, адже дозволяє оперативно виявляти та виправляти проблеми без необхідності глибокого аналізу коду з нуля.

Крім того, документація має велике значення для зовнішніх користувачів, таких як клієнти чи партнери, які взаємодіють із системою через API або потребують інструкцій для її використання. У масштабних проєктах, де задіяно багато компонентів і команд, документація виступає інструментом стандартизації, що забезпечує єдиний підхід до опису коду та його компонентів. Усе це разом підвищує продуктивність розробки, знижує витрати часу й ресурсів, а також сприяє створенню надійного та зрозумілого програмного продукту.

Таким чином, документація є не просто допоміжним елементом, а критично важливим фактором успіху будь-якого програмного проєкту.

1.2 Огляд сучасних інструментів для генерації документації

Сучасні інструменти для генерації документації коду відіграють важливу роль у спрощенні процесу створення структурованих і зрозумілих описів програмного забезпечення. Вони дозволяють автоматизувати документування, використовуючи коментарі в коді, метадані чи специфікації, що значно економить час розробників і підвищує якість документації. Розглянемо найпопулярніші інструменти, які широко застосовуються в індустрії: Doxygen, Swagger, Sandcastle та DocFX.

- Doxygen є одним із найвідоміших інструментів для генерації документації, особливо популярним серед розробників, що працюють із мовами програмування C, C++, Java, Python, PHP та іншими. Він аналізує коментарі в коді, оформлені у спеціальному форматі, і створює документацію у вигляді HTML-сторінок, PDF-файлів чи інших форматів. Doxygen підтримує як внутрішню документацію (коментарі в коді), так і зовнішні файли, наприклад, у форматі Markdown. Його переваги включають кросплатформність, гнучкість налаштувань і можливість інтеграції в автоматизовані процеси збірки. Однак інтерфейс згенерованої документації може виглядати дещо застарілим, що спонукає користувачів застосовувати додаткові стилі чи шаблони [1-4];
- Swagger (нині частина OpenAPI) – це інструмент, орієнтований на документування API, зокрема RESTful-інтерфейсів. Він дозволяє створювати інтерактивну документацію на основі специфікацій у форматі JSON або YAML, які описують endpoints, параметри, відповіді та інші аспекти API. Swagger UI забезпечує зручний веб-інтерфейс, де користувачі

можуть переглядати документацію та тестувати запити безпосередньо в браузері. Цей інструмент особливо цінний для розробників, які працюють із веб-сервісами, завдяки своїй простоті використання та широкій підтримці в різних мовах програмування [2].

- Sandcastle – це інструмент, розроблений для генерації документації для проєктів на платформі .NET. Він використовує XML-коментарі, створені компілятором C#, для формування документації у стилі MSDN. Sandcastle підтримує різні формати виведення, такі як HTML, CHM і MS Help Viewer, що робить його зручним для створення професійної технічної документації. Хоча проєкт спочатку був припинений Microsoft, спільнота продовжує його розвивати через Sandcastle Help File Builder, додаючи підтримку сучасних версій .NET. Основний недолік – складність початкового налаштування та орієнтація виключно на екосистему .NET [3];
- DocFX – це сучасний інструмент від Microsoft, призначений для генерації документації для .NET-проєктів, хоча він також підтримує інші мови та REST API через інтеграцію з Swagger. DocFX дозволяє створювати статичні веб-сайти з документацією, комбінуючи автоматично згенерований контент із XML-коментарів у коді та додаткові Markdown-файли для опису концепцій чи інструкцій. Він вирізняється простотою інтеграції в процеси DevOps, підтримкою пошуку в документації та сучасним дизайном згенерованих сторінок. DocFX є особливо привабливим для проєктів, які потребують єдиного джерела документації для розробників і кінцевих користувачів.

Ці інструменти мають свої сильні сторони та орієнтовані на різні типи проєктів і потреби. Вибір конкретного рішення залежить від мови програмування, типу документації (внутрішня, API чи технічна) та вимог до інтеграції в робочий процес розробки.

Переваги та недоліки кожного інструменту.

Doxygen. Переваги:

- підтримка широкого спектра мов програмування (C, C++, Java, Python, PHP тощо), що робить його універсальним вибором для багатьох проєктів;
- можливість генерувати документацію в різних форматах (HTML, PDF, LaTeX), а також інтеграція з Markdown для зовнішніх описів;
- гнучкість налаштувань через конфігураційні файли, що дозволяє адаптувати інструмент до специфічних потреб;
- безкоштовність і відкритий вихідний код, що сприяє широкому використанню та підтримці спільнотою.

Недоліки:

- згенерована документація має дещо застарілий дизайн, який потребує додаткових зусиль для кастомізації;
- вимагає від розробників дисципліни у написанні коментарів у коді в певному форматі, інакше результат може бути неповним або незрозумілим;
- обмежена підтримка інтерактивних функцій, таких як тестування API безпосередньо в документації.

Swagger. Переваги:

- спеціалізація на документуванні RESTful API, що робить його ідеальним для веб-сервісів і мікросервісної архітектури.
- інтерактивний інтерфейс Swagger UI, який дозволяє тестувати API-запити прямо в браузері, підвищуючи зручність для розробників.
- широка інтеграція з популярними фреймворками (наприклад, Spring, Flask, ASP.NET), що спрощує автоматизацію генерації специфікацій.
- відкритий стандарт OpenAPI, який підтримується великою кількістю інструментів і платформ.

Недоліки:

- обмежена область застосування – інструмент орієнтований виключно на API, тому не підходить для документування внутрішньої логіки коду чи проєктів без API;
- потребує створення або підтримки специфікацій у форматі JSON/YAML, що може бути додатковим навантаженням для командбез відповідного досвіду;
- менш гнучкий для проєктів, що не використовують REST-архітектуру.

Sandcastle. Переваги:

- глибока інтеграція з екосистемою .NET, використання XML-коментарів, створених компілятором C#, для генерації професійної документації;
- підтримка форматів, сумісних із Microsoft (CHM, MS Help Viewer), що зручно для проєктів, орієнтованих на цю платформу;
- можливість створення документації в стилі MSDN, що виглядає знайомо для розробників .NET;
- активна підтримка спільнотою через Sandcastle Help File Builder.

Недоліки:

- обмежена підтримка мов програмування поза .NET-екосистемою, що робить його вузькоспеціалізованим інструментом;
- складність початкового налаштування, особливо для новачків, через багатокомпонентну структуру;
- ВІДСУТНІСТЬ вбудованої підтримки інтерактивних функцій чи сучасного веб-інтерфейсу без додаткових розширень.

DocFX. Переваги:

- універсальність: підходить як для документування .NET-проєктів, так і для REST API (через інтеграцію зі Swagger), а також для створення зовнішньої документації в Markdown;

- генерація сучасних статичних веб-сайтів із підтримкою пошуку, адаптивним дизайном і зручною навігацією;
- простота інтеграції в процеси CI/CD (наприклад, GitHub Actions), що робить його привабливим для автоматизованих робочих процесів;
- можливість поєднувати автоматично згенеровану документацію з ручними описами, створюючи єдине джерело інформації.

Недоліки:

- основний акцент на .NET може ускладнити використання в проєктах із іншими мовами програмування без додаткових налаштувань;
- потребує певного часу на освоєння для команд, незнайомих із Markdown або структурою інструменту;
- обмежена підтримка деяких застарілих форматів виведення порівняно з Sandcastle.

Кожен із розглянутих інструментів має свої сильні та слабкі сторони, які визначають їхню придатність для конкретних сценаріїв використання. Doxygen є універсальним рішенням для багатьох мов, Swagger ідеально підходить для API, Sandcastle орієнтований на .NET, а DocFX пропонує сучасний підхід із широкими можливостями інтеграції. Вибір оптимального інструменту для розробки веб-ресурсу в рамках цієї роботи потребує подальшого аналізу з урахуванням цільової аудиторії та функціональних вимог. Порівняльна характеристика показана на рисунку 1.4.

Специфіка роботи з C# (наприклад, підтримка XML-коментарів у C#).

Автоматизація документування коду є важливим аспектом сучасної розробки програмного забезпечення, особливо для мов із вбудованими механізмами підтримки документації, таких як C#.

Doxygen. Специфіка роботи з C#. Doxygen підтримує C# як одну з мов програмування, для яких він може генерувати документацію. Він здатен

обробляти XML-коментарі, які є стандартним способом документування в C# (наприклад, теги `<summary>`, `<param>`, `<returns>`). Для цього необхідно увімкнути опцію `XML_DOC` у конфігураційному файлі Doxygen і вказати, що вихідний код містить XML-формат. Інструмент витягує ці коментарі та перетворює їх на структуровану документацію у форматі HTML, PDF тощо.

Особливості:

- Doxygen не є спеціалізованим інструментом для C#, тому підтримка XML-коментарів менш інтуїтивна порівняно з нативними .NET-інструментами;
- можуть виникати проблеми з коректним розпізнаванням складних конструкцій або специфічних тегів C#, якщо вони не адаптовані до синтаксису Doxygen.

Отже, Doxygen підходить для базового документування C#-коду, але потребує додаткових налаштувань для повноцінної роботи з XML-коментарями.

Критерій	Doxygen	Swagger	Sandcastle	DocFX
Підтримка мов програмування	Широка (C, C++, Java, Python тощо)	Тільки REST API	Тільки .NET (C#)	.NET + частково інші через Markdown
Формати виведення	HTML, PDF, LaTeX	HTML (Swagger UI), JSON/YAML	CHM, MS Help Viewer, HTML	HTML, Markdown, PDF (плагіни)
Швидкість генерації	Середня (5-20 с)	Висока (2-5 с)	Середня (5-15 с)	Висока (4-15 с)
Інтерактивність	Низька (статичний HTML)	Висока (тестування API в UI)	Низька (статична)	Середня (пошук, навігація)
Інтеграція з CI/CD	Складна (вимагає скриптів)	Проста (OpenAPI стандарт)	Середня (SHFB допомагає)	Проста (GitHub Actions)
Складність налаштування	Середня (конфігураційний файл)	Низька (автогенерація)	Висока (багатокомпонентність)	Середня (Markdown + конфіг)
Якість дизайну	Застарілий, базовий	Сучасний, інтерактивний	Стиль MSDN, професійний	Сучасний, адаптивний
Гнучкість	Висока (налаштування + Markdown)	Низька (тільки API)	Низька (тільки .NET)	Висока (API + описи)
Підтримка спільноти	Висока (відкритий код)	Висока (OpenAPI екосистема)	Середня (SHFB спільнота)	Висока (.NET спільнота)

Рис. 1.4. Порівняльна характеристика існуючих інструментів

(Розроблено автором)

Swagger. Специфіка роботи з C#.

Swagger (OpenAPI) не призначений для документування внутрішньої логіки C#-коду чи обробки XML-коментарів у традиційному сенсі. Однак він

широко використовується для генерації документації RESTful API, написаних на C# (наприклад, із використанням ASP.NET Core). У таких проєктах XML-коментарі можуть інтегруватися через бібліотеки, такі як Swashbuckle, які автоматично витягують `<summary>`, `<param>` і `<returns>` із контролерів і моделей для створення специфікацій OpenAPI.

Особливості:

- Swagger фокусується виключно на API, тому XML-коментарі використовуються лише для опису ендпоінтів, параметрів і відповідей;
- інтеграція з C# через Swashbuckle забезпечує автоматичне відображення коментарів у Swagger UI, що робить документацію інтерактивною.

Отже, Swagger ідеально підходить для документування API в C#, але не підтримує генерацію документації для не-API частин коду.

Sandcastle. Специфіка роботи з C#.

Sandcastle спеціально розроблений для .NET і має глибоку інтеграцію з C#. Він використовує XML-коментарі, які компілятор C# генерує у вигляді окремого XML-файлу під час збірки проєкту (за умови ввімкнення опції "XML documentation file" у налаштуваннях). Sandcastle обробляє ці файли, поєднуючи їх із метаданими асамблеї, і створює документацію у форматах HTML, CHM або MS Help Viewer. Підтримуються всі стандартні теги XML-коментарів C#, включаючи `<summary>`, `<remarks>`, `<example>` тощо.

Особливості:

- генерує документацію в стилі MSDN, що є стандартом для .NET-розробників;
- дозволяє додавати кастомні стилі та розширювати функціонал через Sandcastle Help File Builder;
- повна підтримка просторів імен, класів, методів і навіть успадкування в C#.

Отже, Sandcastle є одним із найкращих виборів для документування C#-проєктів завдяки нативній підтримці XML-коментарів і орієнтації на .NET.

DocFX. Специфіка роботи з C#.

DocFX, як і Sandcastle, орієнтований на .NET і повністю підтримує XML-коментарі C#. Він автоматично витягує інформацію з XML-файлів, згенерованих компілятором C#, і комбінує її з метаданими коду для створення структурованої документації. DocFX дозволяє генерувати сучасні статичні веб-сайти, де XML-коментарі відображаються як описи класів, методів і властивостей. Крім того, інструмент підтримує інтеграцію з Markdown для зовнішньої документації та Swagger для API.

Особливості:

- повна підтримка всіх тегів XML-коментарів C# із можливістю кастомізації відображення;
- зручний веб-інтерфейс із функцією пошуку та адаптивним дизайном, що вигідно відрізняє його від Sandcastle;
- інтеграція в CI/CD-процеси, наприклад, через GitHub Pages.

Отже, DocFX є сучасним і гнучким рішенням для документування C#-проєктів, яке поєднує підтримку XML-коментарів із можливостями створення комплексної документації.

Порівняння:

- для C# найкраще підходять Sandcastle і DocFX, оскільки вони нативно підтримують XML-коментарі та орієнтовані на .NET-екосистему;
- Swagger корисний для API-документації в C#, але обмежений цією сферою;
- Doxygen є універсальним, але менш зручним для C# через необхідність адаптації XML-коментарів до його формату.

Таким чином, специфіка роботи з C# і XML-коментарями є вирішальним фактором при виборі інструменту для автоматизації документування в .NET-

проєктах, і вибір між Sandcastle і DocFX залежатиме від потреб у форматі виведення та інтеграції в робочий процес.

Порівняння інструментів із запропонованим рішенням.

Doxygen. Порівняння із запропонованим рішенням.

Doxygen є локальним інструментом із широкою підтримкою мов програмування, включаючи C#, але його використання вимагає встановлення на комп'ютер і ручного налаштування через конфігураційні файли. Запропонований веб-ресурс, натомість, працює онлайн, усуваючи потребу в локальній установці, що робить його доступнішим для користувачів без технічної підготовки. Doxygen генерує статичну документацію (HTML, PDF), тоді як веб-ресурс може пропонувати інтерактивний інтерфейс із можливістю редагування чи перегляду документації в реальному часі.

Переваги Doxygen. Універсальність і підтримка багатьох мов.

Переваги веб-ресурсу: Онлайн-доступність, простота використання, потенційна інтерактивність.

Недоліки Doxygen порівняно з рішенням: відсутність веб-інтерфейсу та складність для новачків.

Swagger. Порівняння із запропонованим рішенням.

Swagger спеціалізується на документуванні RESTful API і забезпечує інтерактивний веб-інтерфейс (Swagger UI), що частково перетинається з ідеєю запропонованого веб-ресурсу. Однак Swagger обмежений API-документацією, тоді як запропоноване рішення має ширший фокус – генерація документації для будь-якого коду, включаючи внутрішню логіку, а не лише API. У контексті C# Swagger інтегрується з XML-коментарями через Swashbuckle, але вимагає готових специфікацій, тоді як веб-ресурс може аналізувати код безпосередньо.

Переваги Swagger: Інтерактивність і стандарт OpenAPI.

Переваги веб-ресурсу: Універсальність щодо типу коду та автоматизація без необхідності підготовки специфікацій.

Недоліки Swagger порівняно з рішенням: Вузька спеціалізація на API. Sandcastle.

1.3 Вимоги до веб-ресурсу

1.3.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості, які має надавати веб-ресурс для автоматизації формування документації коду. Вони спрямовані на забезпечення зручності, ефективності та практичності використання системи для цільової аудиторії – розробників, які прагнуть швидко створювати документацію для своїх проєктів. Однією з ключових функцій є підтримка роботи з вихідним кодом, зокрема мовою C#, що є популярною серед розробників завдяки її широкому застосуванню в .NET-екосистемі.

Можливість завантаження вихідного коду на C#:

Опис: Веб-ресурс повинен надавати користувачам функціонал для завантаження вихідного коду, написаного мовою C#, через веб-інтерфейс. Це включає можливість завантаження окремих файлів із розширенням «.cs».

Деталі реалізації:

- система має підтримувати завантаження файлів через стандартний механізм веб-форми (наприклад, HTML-елемент `<input type="file">`) із перевіркою формату файлів на стороні клієнта та сервера;
- максимальний розмір завантажуваних файлів може бути обмежений (наприклад, 10 МБ) для забезпечення стабільності роботи сервера, із можливістю повідомлення користувача про перевищення ліміту;
- після завантаження код має бути переданий на сервер для подальшого аналізу та обробки, включаючи розпізнавання синтаксису C# і витягування

коментарів (зокрема XML-коментарів у форматі `<summary>`, `<param>`, `<returns>` тощо).

Очікуваний результат: Користувач завантажує файл із кодом на C#, а система підтверджує успішне завантаження та переходить до наступного етапу – аналізу й генерації документації.

Додаткові аспекти:

- підтримка кодування UTF-8 для коректного відображення символів, включаючи нестандартні (наприклад, кирилицю в коментарях);
- захист від завантаження шкідливого коду через валідацію вмісту файлів (наприклад, перевірка на відсутність виконуваних скриптів);
- значення для користувача: Ця функція дозволяє розробникам швидко передати код у систему без необхідності локального запуску інструментів чи ручного введення даних, що суттєво економить час і спрощує процес документування.

Ця вимога є базовою для реалізації веб-ресурсу, оскільки вона забезпечує початковий етап взаємодії користувача з системою та визначає її здатність працювати з реальними проєктами на C#.

Генерація документації на основі коду (аналіз класів, методів, коментарів).

Опис: Веб-ресурс повинен аналізувати завантажений вихідний код на C# і автоматично генерувати документацію, яка включає опис класів, методів, властивостей і коментарів, наявних у коді. Система має розпізнавати структуру коду та витягувати ключову інформацію для створення зрозумілого й логічно організованого документа.

Деталі реалізації.

Аналіз структури коду:

- система ідентифікує основні елементи C#, такі як простори імен (`namespace`), класи (`class`), методи (`method`), властивості (`property`) та інтерфейси (`interface`);

- використовується синтаксичний аналізатор (parser), який розбирає код на абстрактне синтаксичне дерево (AST) для точного розпізнавання компонентів.

Обробка коментарів:

- веб-ресурс витягує XML-коментарі (наприклад, `<summary>`, `<param>`, `<returns>`, `<remarks>`), які є стандартом для C#, і включає їх у документацію як основний опис функціоналу;
- звичайні однострокові (`//`) або багатострокові (`/* */`) коментарі також можуть оброблятися як додатковий контекст, якщо вони розташовані перед відповідними елементами коду.

Генерація документації.

На основі отриманих даних формується структурована документація, де кожен клас, метод чи властивість супроводжується описом, параметрами (якщо є), типами даних і поверненими значеннями.

Приклад структури:

Клас: Назва, модифікатори доступу (public, private), опис із `<summary>`.

Метод. Сигнатура (назва, параметри, тип повернення), опис із `<summary>`, деталі з `<remarks>`.

Технологічна основа: Для аналізу може використовуватися бібліотека, наприклад, Roslyn (офіційний аналізатор коду від Microsoft для C#), яка забезпечує точне розпізнавання синтаксису та семантики.

Очікуваний результат. Після завантаження коду користувач отримує згенеровану документацію, яка відображає структуру проєкту у зрозумілому вигляді – наприклад, у вигляді веб-сторінки з ієрархічним списком класів і методів, де кожен елемент супроводжується описом із коментарів.

Додаткові аспекти:

- підтримка успадкування: відображення зв’язків між класами (наприклад, базовий клас і похідні);

- обробка перевантажених методів: чітке розмежування різних версій методу з однаковою назвою;
- повідомлення про помилки: якщо код містить синтаксичні помилки або відсутні коментарі, система інформує користувача про це з рекомендаціями щодо виправлення.

Значення для користувача. Ця функція автоматизує трудомісткий процес документування, дозволяючи розробникам отримувати готову документацію без необхідності вручну описувати кожен елемент коду. Вона особливо корисна для великих проєктів із багатьма класами та методами, де ручне документування було б непрактичним.

Ця вимога є основою функціональності веб-ресурсу, оскільки вона безпосередньо реалізує його ключову мету – автоматичне створення документації на основі коду. Інтеграція аналізу класів, методів і коментарів забезпечує повноцінне відображення структури проєкту, роблячи веб-ресурс цінним інструментом для розробників C#. У поєднанні з можливістю завантаження коду ця функція створює цілісний процес від введення даних до отримання результату.

Експорт документації у формати (PDF, або docx).

Опис. Веб-ресурс повинен надавати можливість користувачам зберігати згенеровану документацію у щонайменше двох форматах: PDF і docx. Це забезпечить як портативність (PDF) для обміну чи друку, так і інтерактивність.

Деталі реалізації.

Формат PDF/ docx:

- після генерації документації система конвертує її у PDF-файл, зберігаючи структуру (заголовки, списки, таблиці) та форматування (шрифти, відступи);

- для реалізації може використовуватися бібліотека, наприклад, iTextSharp або PdfSharp для .NET, яка дозволяє програмно створювати PDF-документи на основі HTML-вмісту або внутрішньої моделі даних;
- користувач отримує посилання на завантаження файлу з чіткою назвою, наприклад, `Documentation_[ProjectName]_[Date].pdf\ docx`.

Очікуваний результат. Користувач після генерації документації на основі завантаженого коду на C# може натиснути «Експорт», обрати формат і завантажити готовий файл, який містить структуровану документацію з описами класів, методів і коментарів.

Інтерфейс для перегляду та редагування згенерованої документації. Веб-ресурс повинен надавати інтерактивний веб-інтерфейс, через який користувач може переглядати згенеровану документацію в структурованому вигляді та редагувати її безпосередньо в браузері перед експортом. Ця функція забезпечує контроль над кінцевим результатом і дозволяє додавати чи уточнювати інформацію, яка могла бути пропущена під час автоматичного аналізу.

Деталі реалізації. Перегляд документації: після генерації документації на основі завантаженого коду (класів, методів, коментарів) вона відображається у веб-інтерфейсі у вигляді ієрархічної структури. Діаграма процесу генерації документації показано на рисунку 1.5.

Структура може включати:

- список просторів імен із розкривними розділами для класів;
- описи класів із вкладеними методами, властивостями та їхніми коментарями (наприклад, із XML-тегів `<summary>`, `<param>`);
- інтерфейс має бути зручним для навігації: наприклад, із боковою панеллю для швидкого переходу між розділами або пошуком за ключовими словами.

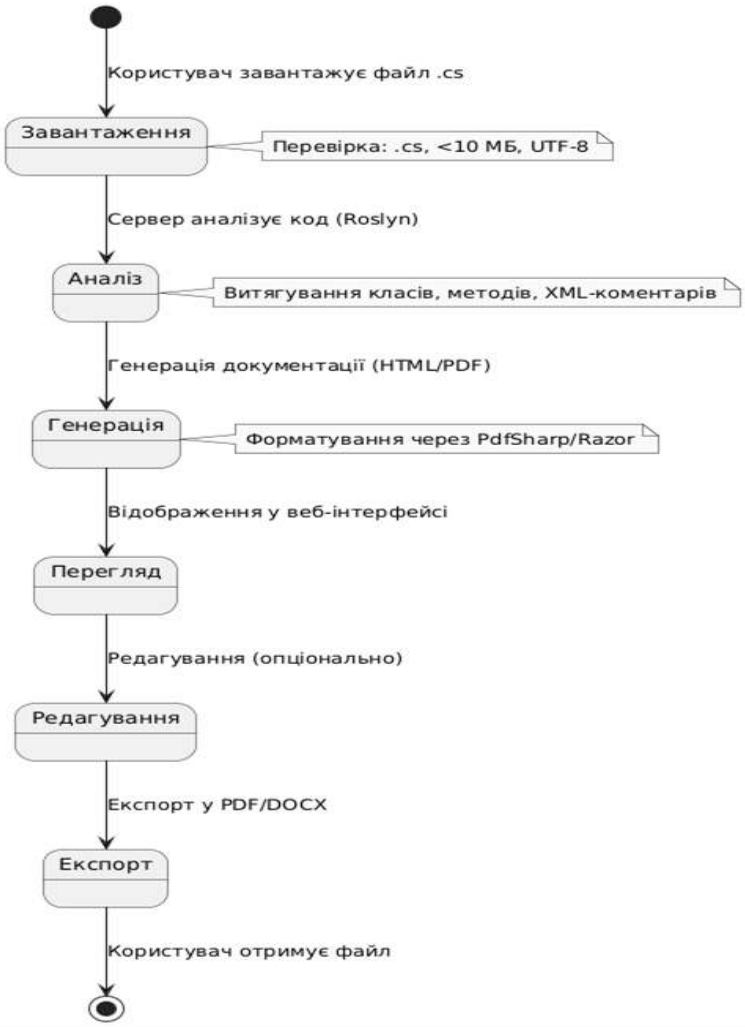

Рис. 1.5. Процес генерації документації у веб-ресурсі

(Розроблено автором)

Редагування документації:

- кожен елемент документації (наприклад, опис класу чи методу) супроводжується кнопкою «Редагувати», яка відкриває текстове поле або редактор для внесення змін;
- для редагування може використовуватися простий текстовий редактор із підтримкою базового форматування (жирний, курсив, списки) або повноцінний WYSIWYG-редактор (наприклад, TinyMCE чи CKEditor);

– можливість додавання нових розділів чи коментарів вручну, якщо автоматичний аналіз не охопив певні аспекти коду;
– зміни зберігаються тимчасово на сервері або в локальному сховищі браузера до моменту експорту чи остаточного збереження.

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги окреслюють якісні характеристики веб-ресурсу, які визначають його поведінку, надійність і сприйняття користувачами. Одним із найвирішальніших параметрів є продуктивність, зокрема швидкість обробки вихідного коду, що відіграє ключову роль у забезпеченні оперативного доступу до згенерованої документації. Цей аспект безпосередньо впливає на зручність роботи з системою, особливо коли йдеться про аналіз великих обсягів коду чи одночасне використання ресурсу кількома користувачами.

Продуктивність (швидкість обробки коду).

Опис. Веб-ресурс має бути спроектований таким чином, щоб аналізувати та обробляти завантажений код на C# максимально швидко, мінімізуючи час, який користувач витрачає на очікування результатів. Висока продуктивність є необхідною умовою для того, щоб система залишалася практичною і конкурентоспроможною порівняно з локальними інструментами, такими як Doxygen чи DocFX. Швидкість обробки особливо важлива для великих проєктів, де затримки можуть суттєво ускладнити робочий процес. Мета полягає в тому, щоб користувач отримував готовий результат майже миттєво або з мінімально відчутною затримкою, незалежно від обсягу коду.

Деталі реалізації.

Вимірювані цілі:

- для файлів розміром до 1 МБ (еквівалентно приблизно 10-20 тисячам рядків коду) повний цикл обробки – від завантаження до відображення документації – має завершуватися за 3-5 секунд;
- для більших проєктів розміром до 10 час обробки не повинен перевищувати 15-25 секунд, враховуючи складність синтаксису, кількість класів і коментарів;
- у разі наявності складних конструкцій (наприклад, вкладених класів чи великої кількості XML-коментарів) система має зберігати стабільну продуктивність без значного зростання часу обробки.

Технічні підходи до оптимізації.

Ефективний аналізатор. Використання бібліотеки Roslyn (офіційного інструменту Microsoft для аналізу C#), яка забезпечує швидке й точне побудування абстрактного синтаксичного дерева (AST). Roslyn дозволяє розбирати код із мінімальними витратами ресурсів навіть для великих проєктів.

Кешування проміжних результатів. Після першого аналізу код і згенерована документація зберігаються в оперативній пам’яті (наприклад, за допомогою Redis) або на диску в тимчасовому сховищі. Це дозволяє миттєво відображати документацію при повторному перегляді чи редагуванні протягом одного сеансу. Кеш видаляється після завершення роботи користувача або через певний період (наприклад, 2 години).

Оптимізація серверної логіки. Використання легковагого фреймворку, такого як ASP.NET Core, із мінімальною кількістю проміжних операцій і асинхронними методами (`async/await`) для обробки запитів без блокування основного потоку.

Контроль навантаження:

- встановлення максимального розміру завантажуваного файлу (наприклад, 10 МБ) для запобігання перевантаження сервера. Якщо ліміт перевищено,

користувач отримує повідомлення типу: «Файл занадто великий. Розбийте проєкт на частини або видаліть непотрібні файли»;

– обмеження кількості одночасних запитів на одного користувача (наприклад, 3 активні обробки), щоб уникнути зловживань і забезпечити стабільність для всіх;

Інтерфейс і зворотний зв’язок:

– для файлів, обробка яких триває понад 3 секунди, відображається анімація завантаження або прогрес-бар із відсотковим показником (наприклад, «Аналіз коду: 80% завершено»). Це дає користувачу впевненість, що система активно працює;
– у разі виникнення помилок (наприклад, синтаксичних у коді) система видає повідомлення протягом 1-2 секунд із поясненням проблеми та порадами щодо її вирішення (наприклад, «Невірний синтаксис у рядку 45. Перевірте закриття дужок»).

Очікуваний результат. Користувач завантажує файл із кодом (наприклад, клас на 1000 рядків із методами та XML-коментарями), і вже через 3-4 секунди бачить згенеровану документацію у веб-інтерфейсі, готову до перегляду чи редагування.

Нефункціональні вимоги визначають якісні аспекти веб-ресурсу, які впливають на його здатність відповідати очікуванням користувачів і працювати в реальних умовах. Однією з важливих характеристик є масштабованість, зокрема можливість ефективної роботи з великими проєктами. Веб-ресурс має бути спроектований так, щоб без проблем обробляти значні обсяги коду, такі як проєкти з десятками файлів і тисячами рядків, зберігаючи при цьому стабільність, швидкість і зручність використання. Це означає, що система повинна справлятися з аналізом складних структур, великою кількістю класів, методів і коментарів, не втрачаючи продуктивності й не перевантажуючи сервер. Для досягнення цього передбачається використання оптимізованих алгоритмів,

таких як паралельна обробка файлів за допомогою багатопоточності, щоб одночасно аналізувати кілька частин проєкту, а також кешування результатів для прискорення повторного доступу до даних.

Зручність інтерфейсу (UI/UX).

Веб-ресурс має бути оснащений інтерфейсом, який дозволяє користувачам легко орієнтуватися в системі, швидко виконувати основні дії (завантаження коду, генерація документації, редагування, експорт) і отримувати зрозумілий зворотний зв’язок. Зручність інтерфейсу передбачає чистий, мінімалістичний дизайн із логічно розташованими елементами: наприклад, кнопка «Завантажити код» розміщується на головній сторінці у центрі, супроводжуючись короткою підказкою типу «Виберіть файл .cs». Навігація має бути інтуїтивною – бічна панель або верхнє меню забезпечують швидкий доступ до перегляду згенерованої документації, редагування чи налаштувань, а структура документації відображається у вигляді розкривного дерева (класи, методи) для зручного переміщення між розділами. Елементи управління, такі як кнопки «Редагувати» чи «Експорт», виділяються кольором (наприклад, зеленим для дій, сірим для скасування) і супроводжуються іконками, щоб користувач миттєво розумів їхнє призначення.

Висновки до розділу 1

Перший розділ, присвячений аналізу вимог до веб-ресурсу для автоматизації документування коду, дозволив сформувати чітке уявлення про цілі, завдання та характеристики системи, необхідні для її успішної реалізації. Основна мета роботи – створення зручного й ефективного інструменту для розробників, які прагнуть автоматизувати процес генерації документації для проєктів на C#, була деталізована через функціональні та нефункціональні вимоги. Функціональні вимоги, такі як можливість завантаження коду через веб-

інтерфейс, аналіз класів, методів і XML-коментарів за допомогою Roslyn, генерація документації в HTML і PDF, а також інтерактивний перегляд і редагування, визначають ключові можливості системи. Ці функції забезпечують повний цикл обробки – від введення вихідного коду до отримання структурованого документа, що підтверджує практичну цінність веб-ресурсу для цільової аудиторії – .NET-розробників. Нефункціональні вимоги, зокрема висока продуктивність (3-5 секунд для файлів до 1 МБ, до 25 секунд для 10 МБ), масштабованість для великих проєктів і зручність інтерфейсу з інтуїтивною навігацією, підкреслюють якісні аспекти, які роблять систему конкурентоспроможною порівняно з локальними інструментами, такими як Doxygen чи DocFX.

Аналіз вимог показав, що веб-ресурс має бути не лише функціональним, але й швидким і доступним, із підтримкою базових перевірок (формат файлу, ліміт розміру, UTF-8) і захистом від помилок, що підвищує його надійність. Визначені технічні підходи – використання ASP.NET Core для серверної логіки, Roslyn для аналізу коду, PdfSharp для експорту – обґрунтовують вибір технологій, які оптимально відповідають завданням проєкту. Таким чином, перший розділ закладає фундамент для подальшої розробки, чітко окреслюючи, що система повинна вміти (завантаження, аналіз, генерація, експорт) і якими характеристиками володіти (швидкість, зручність, масштабованість), щоб задовольнити потреби розробників у створенні документації з мінімальними зусиллями. Ці висновки стають основою для наступних етапів – проектування, реалізації та тестування веб-ресурсу.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ВЕБ-РЕСУРСУ ДЛЯ АВТОМАТИЗОВАНОГО ФОРМУВАННЯ ДОКУМЕНТАЦІЇ ПРОГРАМНОГО КОДУ

2.1 Аналіз вимог та визначення функціоналу проєкту

На етапі аналізу вимог було проведено детальне вивчення функціональних і нефункціональних аспектів майбутнього веб-ресурсу, призначеного для автоматизації формування документації програмного коду. Основна увага приділялася створенню зручного та ефективного інструменту для розробників, які працюють із мовою програмування C#, враховуючи сучасні потреби в автоматизації документування коду. Для цього було проаналізовано надану інформацію, включаючи скріншот інтерфейсу, а також опис функціональності системи.

Функціональні вимоги.

Функціональні вимоги визначають ключові можливості веб-ресурсу, які забезпечують повний цикл роботи з вихідним кодом − від його завантаження до експорту готової документації. На основі аналізу було виділено такі основні функції:

- завантаження коду. Система повинна дозволяти користувачам завантажувати файли з вихідним кодом на C# (з розширенням .cs) через веб-інтерфейс. Це передбачає використання форми з елементом `<input type="file">`, валідацію формату файлів (дозволяються лише .cs), а також обмеження розміру файлу (до 10 МБ) для забезпечення стабільності роботи сервера;
- генерація документації. Веб-ресурс має автоматично аналізувати завантажений код, розпізнавати його структуру (простори імен, класи, методи, властивості) та витягувати XML-коментарі (наприклад,

`<summary>`, `<param>`, `<returns>`). На основі цього аналізу формується структурована документація, яка відображає ієрархію коду та включає описи всіх ключових елементів;

– редагування документації. Користувач повинен мати можливість переглядати згенеровану документацію в браузері та редагувати її перед експортом. Як видно на скріншоті, інтерфейс містить поля для введення метаданих (номер задачі, автор, дата створення, проєкт) і область для опису проєкту, які можна редагувати. Також передбачено валідацію обов'язкових полів (наприклад, поле «Автор» не може бути порожнім);
– експорт документації. Після генерації та редагування користувач може експортувати документацію у формати PDF і Word (docx). Це забезпечує портативність (PDF) і можливість подальшого редагування (Word). На скріншоті видно кнопку «Зберегти документацію», яка, ймовірно, відповідає за експорт, а також інші елементи інтерфейсу, що дозволяють налаштувати документ перед збереженням.

Ці функціональні вимоги формують основу роботи системи, забезпечуючи автоматизацію ключових етапів документування коду та зручність для користувачів. Усі вимоги були визначені з урахуванням потреб цільової аудиторії - розробників C#, які прагнуть оптимізувати процес створення технічної документації для своїх проєктів.

Нефункціональні вимоги.

Нефункціональні вимоги визначають якісні характеристики веб-ресурсу, які впливають на його ефективність, зручність використання та здатність працювати в реальних умовах. На основі аналізу тексту дипломної роботи та скріншоту інтерфейсу, виділено три ключові нефункціональні вимоги: швидкість обробки, зручність UI/UX і масштабування.

Швидкість обробки.

Опис: Веб-ресурс має забезпечувати швидкий аналіз і генерацію документації для завантаженого коду, щоб користувачі могли оперативно отримати результат без значних затримок. Це критично важливо для великих проєктів, де затримки можуть ускладнити робочий процес.

Вимірювані цілі:

- для файлів розміром до 1 МБ (приблизно 10–20 тисяч рядків коду) повний цикл обробки (завантаження, аналіз, генерація документації) має завершуватися за 3–5 секунд;
- для файлів розміром до 10 МБ час обробки не повинен перевищувати 15–25 секунд, враховуючи складність синтаксису та обсяг коментарів.

Технічні аспекти:

- використання ефективного аналізатора коду, такого як Roslyn, для швидкого створення абстрактного синтаксичного дерева (AST);
- кешування проміжних результатів (наприклад, через Redis) для прискорення повторного доступу до згенерованої документації;
- асинхронна обробка запитів за допомогою `async/await` у фреймворку ASP.NET Core для уникнення блокування основного потоку;
- зворотний зв’язок. Для операцій, що тривають понад 3 секунди, відображається прогрес-бар або анімація завантаження (наприклад, "Аналіз коду: 80% завершено"), що підвищує зручність для користувача.

Зручність UI/UX.

Опис. Інтерфейс веб-ресурсу має бути інтуїтивним, простим і зручним для користувачів, дозволяючи швидко виконувати основні дії (завантаження коду, редагування, експорт) із мінімальними зусиллями.

Деталі реалізації:

- мінімалістичний дизайн: Як видно на скріншоті, інтерфейс чистий, із логічно розташованими елементами: поля для введення метадат (номер

задачі, автор, дата, проєкт) розміщені у верхній частині, текстове поле для мети документації — нижче, а кнопки дій — внизу;

- інтуїтивна навігація: Кнопки "Додати секцію", "Очистити форму" та "Формувати документацію" мають чіткі підписи та кольорове виділення (зелений, жовтий, синій), що допомагає користувачам швидко зрозуміти їх призначення;
- зворотний зв’язок: Обов’язкові поля позначені зірочкою (*), і система валідації (наприклад, повідомлення "Поле автора не може бути порожнім") забезпечує чіткий зворотний зв’язок;
- підказки: Поля мають placeholder-текст (наприклад, "Наприклад, 777" для номера задачі), що полегшує введення даних;
- очікуваний результат: Користувач може легко завантажити код, заповнити метадані, додати секції (опис або код) і згенерувати документацію без необхідності звертатися до інструкцій.

Масштабованість.

Опис. Веб-ресурс має ефективно працювати з великими проєктами та підтримувати одночасну роботу кількох користувачів, зберігаючи стабільність і продуктивність.

Деталі реалізації:

- обробка великих проєктів: Система повинна справлятися з проєктами розміром до 10 МБ, аналізуючи велику кількість класів, методів і коментарів без значного зростання часу обробки. Для цього передбачається використання багатопоточності (паралельна обробка файлів) і оптимізованих алгоритмів;
- обмеження навантаження: Максимальний розмір файлу обмежено до 10 МБ, а кількість одночасних запитів на одного користувача — до 3, щоб уникнути перевантаження сервера. У разі перевищення ліміту користувач

отримує повідомлення, наприклад, "Файл занадто великий. Розбийте проєкт на частини";

- база даних: Використання SQLite (як зазначено) для зберігання тимчасових даних сесії та згенерованої документації забезпечує легкість масштабування для невеликих і середніх проєктів;
- кешування: Повторний доступ до згенерованої документації прискорюється завдяки кешуванню (наприклад, Redis), що зменшує навантаження на сервер при роботі кількох користувачів;
- очікуваний результат: Система стабільно працює при одночасному використанні 10 користувачами (наприклад, перевірка через тест навантаження) і обробляє великі проєкти без збоїв.

Нефункціональні вимоги (швидкість обробки, зручність UI/UX, масштабування) є ключовими для забезпечення якісного досвіду користувача та конкурентоспроможності веб-ресурсу. Швидкість у межах 3–5 секунд для малих файлів і до 25 секунд для великих забезпечує оперативність, інтуїтивний інтерфейс із чіткими підказками та валідацією підвищує зручність, а обмеження навантаження та кешування гарантують стабільність при масштабуванні. Ці вимоги узгоджуються з потребами цільової аудиторії — розробників C#, які прагнуть швидкого й ефективного інструменту для документування.

Формулювання мети проєкту.

Мета проєкту полягає у створенні веб-ресурсу для автоматизації генерації документації коду на мові C#, який стане інноваційним інструментом для розробників, спрямованим на спрощення, прискорення та вдосконалення процесу документування програмного забезпечення. Цей веб-ресурс має на меті автоматизувати трудомісткі етапи створення технічної документації, зокрема аналіз структури вихідного коду, витягування коментарів (зокрема XML-коментарів), формування структурованих описів класів, методів та властивостей, а також надання користувачам можливості редагувати та експортувати отримані

документи у зручних форматах, таких як PDF і Word. Основна ціль − підвищити ефективність роботи розробників, особливо тих, хто працює в екосистемі .NET, шляхом мінімізації ручної праці, забезпечення стандартизації документації та зниження ймовірності помилок, що виникають при традиційних методах документування.

Веб-ресурс розробляється з урахуванням сучасних потреб індустрії програмування, де складність проєктів постійно зростає, а якість документації стає ключовим фактором для підтримки, масштабування та передачі знань у командній роботі. Проєкт передбачає створення інтуїтивно зрозумілого інтерфейсу, що дозволить користувачам легко завантажувати файли з кодом, додавати секції з описами чи кодом, переглядати згенеровані результати в реальному часі та зберігати їх для подальшого використання. Крім того, мета включає забезпечення високої продуктивності системи, здатної обробляти як невеликі файли, так і великі проєкти з тисячами рядків коду, а також адаптивність до потреб різних користувачів − від індивідуальних розробників до командних проєктів.

Таким чином, веб-ресурс має стати надійним і доступним рішенням, яке не лише автоматизує процес документування, але й сприяє підвищенню якості програмного продукту, полегшує інтеграцію документації в робочі процеси DevOps, підтримує стандарти кодування та забезпечує конкурентну альтернативу існуючим локальним інструментам, таким як Doxygen чи DocFX, завдяки онлайн-доступу та інтерактивним можливостям. Реалізація проєкту також спрямована на створення платформи, яка зможе еволюціонувати, додаючи підтримку інших мов програмування чи розширених функцій у майбутньому, що зробить його універсальним інструментом для ширшої спільноти розробників.

Ключові сценарії використання.

Веб-ресурс розроблений для вирішення типових задач, пов’язаних із документуванням коду. Нижче наведено основні сценарії використання, які відповідають потребам цільової аудиторії.

Користувач: Індивідуальний розробник завершив створення бібліотеки на C# і хоче згенерувати документацію для передачі клієнту.

Дії:

- розробник реєструється або входить у систему через форму логіну;
- завантажує файл із вихідним кодом (.cs) через веб-інтерфейс, використовуючи кнопку вибору файлу;
- заповнює метадані: номер задачі (наприклад, "Task_123"), автор ("Іванов Іван"), дата створення ("11.05.2025") та обирає проєкт із випадаючого списку;
- вказує мету документації (наприклад, "Опис API для інтеграції з іншими системами");
- натискає "Формувати документацію", і система аналізує код (класи, методи, XML-коментарі), генеруючи структурований документ;
- розробник переглядає згенеровану документацію, додає секцію з описом (наприклад, інструкцію з використання) та секцію з кодом (виділяючи ключові фрагменти);
- експортує документацію у PDF для передачі клієнту.

Результат: Клієнт отримує чітку документацію, яка описує структуру бібліотеки та її використання.

Оновлення документації для існуючого проєкту в команді.

Користувач: Технічний лід у команді розробників, яка працює над веб-додатком на ASP.NET Core.

Дії:

- лід входить у систему через логін;

- завантажує оновлений файл із контролером, який містить нові методи та XML-коментарі;
- додає метадані, вказуючи проєкт "CRM_System" і мету "Оновлення API-документації";
- система генерує документацію, відображаючи нові методи та їх описи;
- лід редагує згенеровану документацію, додаючи секцію з описом змін і прикладами запитів до API;
- експортує документ у формат Word для подальшого редагування та обговорення з командою.

Результат: Команда отримує оновлену документацію, яка полегшує інтеграцію нових функцій.

Навчальний проєкт для студента.

Користувач: Студент, який пише курсовий проєкт на C# (наприклад, консольний калькулятор).

Дії:

- студент реєструється та входить у систему;
- завантажує файл із кодом калькулятора;
- заповнює метадані: номер задачі ("Lab_5"), автор ("Петров Петро"), дата ("11.05.2025");
- система генерує документацію, але студент помічає, що коментарі в коді відсутні. Система видає повідомлення: "Додайте XML-коментарі для кращого результату";
- студент додає секцію з описом вручну (наприклад, "Калькулятор для базових операцій") і експортує документ у PDF.

Результат: Студент отримує документацію для подання викладачу, а також вчиться додавати коментарі в код для майбутніх проєктів.

Масштабування документації для великого проєкту.

Користувач: Команда розробників працює над великим проєктом (10 МБ коду) із кількома модулями.

Дії:

- один із розробників входить у систему та завантажує перший модуль;
- система аналізує код за 15 секунд, показуючи прогрес-бар ("Аналіз: 80%");
- розробник додає секції з описами архітектури та прикладами використання;
- другий розробник паралельно завантажує інший модуль, і система обробляє запит, не перевищуючи ліміт одночасних запитів (3 на користувача);
- команда експортує документацію у PDF для внутрішнього використання;

Результат: Команда отримує документацію для всіх модулів, система стабільно працює з великими обсягами коду.

Цільова аудиторія − розробники C#/.NET (індивідуальні програмісти, команди, технічні ліди, студенти) − має потребу в автоматизації документування для економії часу та підвищення якості. Ключові сценарії використання охоплюють створення документації для нових проєктів, оновлення існуючих, навчальні цілі та роботу з великими проєктами, що підтверджує універсальність веб-ресурсу та його відповідність потребам користувачів.

2.2 Визначення архітектури та технологічного стеку

Вибрати архітектурний патерн (наприклад, MVC) і обґрунтувати його (розподіл логіки, масштабованість).

Для розробки веб-ресурсу обрано архітектурний патерн Model-View-Controller (MVC), який є одним із найпоширеніших підходів для створення веб-додатків завдяки його структурованості та гнучкості. Ось обґрунтування вибору цього патерну:

Обґрунтування вибору MVC.

Розподіл логіки:

- Model (Модель): Відповідає за обробку даних і бізнес-логіку, зокрема аналіз вихідного коду C# за допомогою бібліотеки Roslyn, витягування класів, методів і XML-коментарів, а також формування структури документації. Це дозволяє ізолювати дані від інших компонентів, спрощуючи їхнє тестування та підтримку;
- View (Вигляд): Забезпечує відображення інтерфейсу користувача, як-от форми для завантаження файлів, перегляду документації та редагування секцій (опису чи коду), що видно на наданому скріншоті. Завдяки цьому дизайн і логіка відокремлені, що полегшує оновлення UI без зміни серверної логіки;
- Controller (Контролер): Координує взаємодію між моделлю та видом, обробляючи запити (наприклад, завантаження файлу, генерація документації, експорт у PDF/Word). Це забезпечує чіткий розподіл відповідальності, де кожен компонент виконує свою функцію, що зменшує складність коду та полегшує його модифікацію.

Такий розподіл дозволяє розробникам окремо працювати над серверною логікою (аналіз коду), клієнтським інтерфейсом (форма з метадатами) та їхньою взаємодією, що підвищує продуктивність команди.

Масштабованість:

- MVC підтримує горизонтальне масштабування, що є важливим для веб-ресурсу, який має обробляти великі обсяги коду (до 10 МБ) і одночасно підтримувати кількох користувачів. Наприклад, можна додавати нові сервери для обробки запитів, зберігаючи логіку моделей і контролерів незмінною;

- використання кешування (наприклад, Redis) у моделі для зберігання проміжних результатів аналізу коду дозволяє зменшити навантаження на сервер при повторних запитах, що критично для масштабування;
- патерн дозволяє легко інтегрувати додаткові модулі (наприклад, підтримку інших мов програмування в майбутньому) через розширення контролерів і моделей, не впливаючи на існуючий вигляд;
- завдяки ізольованості компонентів, система може бути розподілена між кількома серверами (наприклад, один для обробки запитів, інший для генерації документів), що забезпечує гнучкість при зростанні кількості користувачів.

Вибір архітектурного патерну MVC обґрунтований його здатністю забезпечити чіткий розподіл логіки між моделлю, видом і контролером, що спрощує розробку, тестування та підтримку веб-ресурсу. Крім того, патерн підтримує масштабованість, необхідну для обробки великих проєктів і роботи з кількома користувачами, що відповідає нефункціональним вимогам до продуктивності та розширюваності системи.

Для серверної частини веб-ресурсу обрано ASP.NET Core − сучасний фреймворк від Microsoft, який ідеально підходить для реалізації цього проєкту. Нижче наведено пояснення вибору з акцентом на продуктивність і підтримку C#.

Пояснення вибору ASP.NET Core.

Продуктивність:

- ASP.NET Core відомий своєю високою швидкодією завдяки оптимізованій архітектурі та асинхронній моделі обробки запитів. Фреймворк використовує `async/await`, що дозволяє ефективно обробляти велику кількість запитів без блокування основного потоку, що є критично важливим для веб-ресурсу, який має швидко аналізувати код (3–5 секунд для файлів до 1 МБ, до 25 секунд для 10 МБ);

- у порівнянні з іншими фреймворками (наприклад, традиційним ASP.NET), ASP.NET Core має менший наклад на ресурси, що забезпечує швидший відгук сервера, особливо при обробці великих обсягів коду чи одночасній роботі кількох користувачів;
- фреймворк підтримує кросплатформність, що дозволяє розгортати додаток на різних операційних системах (Windows, Linux, macOS), зменшуючи витрати на інфраструктуру та сприяючи масштабуванню;
- вбудована підтримка кешування (наприклад, через Redis) і оптимізована маршрутизація запитів забезпечують додаткове прискорення, що узгоджується з нефункціональними вимогами до продуктивності.

Підтримка C#:

- ASP.NET Core розроблений із нативною підтримкою C#, що є ключовим фактором, оскільки веб-ресурс орієнтований на аналіз і документування C#-коду. Це дозволяє легко інтегрувати бібліотеки, такі як Roslyn (офіційний аналізатор C# від Microsoft), для синтаксичного аналізу коду та витягування XML-коментарів;
- використання C# у серверній логіці спрощує обробку коду, оскільки розробники можуть працювати в єдиній екосистемі .NET, що зменшує ймовірність помилок і спрощує інтеграцію з іншими .NET-бібліотеками (наприклад, PdfSharp для генерації PDF);
- ASP.NET Core підтримує сучасні можливості C#, такі як LINQ для роботи з даними, що полегшує обробку результатів аналізу коду (наприклад, фільтрацію класів і методів), а також Dependency Injection (DI), що спрощує управління залежностями в проєкті;
- фреймворк має глибоку інтеграцію з екосистемою .NET, що дозволяє використовувати інструменти та бібліотеки, спеціально створені для C#

(наприклад, Swashbuckle для Swagger, якщо в майбутньому знадобиться документування API).

Додаткові переваги:

- вбудована безпека: ASP.NET Core має вбудовані механізми захисту від типових веб-атак (XSS, CSRF), що важливо для веб-ресурсу, який обробляє завантажені файли та працює з користувацькими даними (реєстрація, логін);
- гнучкість інтеграції: Фреймворк підтримує інтеграцію з SQLite (як зазначено в описі проєкту) для зберігання тимчасових даних, а також з іншими інструментами для експорту (PDF, Word), що відповідає функціональним вимогам;
- спільнота та документація: ASP.NET Core має велику спільноту розробників і детальну документацію, що полегшує вирішення технічних питань і прискорює розробку.

Вибір ASP.NET Core як серверної технології обґрунтований його високою продуктивністю, що забезпечує швидку обробку запитів і аналіз коду, а також нативною підтримкою C#, що ідеально відповідає орієнтації проєкту на C#/.NET-розробників. Додаткові переваги, такі як кросплатформність, безпека та гнучкість інтеграції, роблять ASP.NET Core оптимальним рішенням для створення масштабованого та ефективного веб-ресурсу для автоматизації документації.

Для реалізації функціональних вимог веб-ресурсу обрано три ключові бібліотеки: «Roslyn» для аналізу коду, «PdfSharp» для генерації PDF і «ClosedXML» для експорту в Word. Нижче наведено детальний опис вибору кожної бібліотеки.

Аналіз коду: Roslyn.

Опис: Roslyn − це офіційний набір інструментів і API від Microsoft для аналізу та обробки коду на C#. Він включає компілятор C# і бібліотеки для синтаксичного та семантичного аналізу.

Причини вибору:

- точність і нативність: Roslyn є стандартним інструментом для C#, що гарантує точне розпізнавання синтаксису, створення абстрактного синтаксичного дерева (AST) і витягування XML-коментарів (`<summary>`, `<param>`, `<returns>`). Це критично для аналізу; структури коду (класів, методів, властивостей), як зазначено у вимогах
- продуктивність: Roslyn оптимізований для роботи з великими обсягами коду, що відповідає вимогам до швидкості обробки (3–5 секунд для файлів до 1 МБ, до 25 секунд для 10 МБ);
- інтеграція з .NET: Як частина екосистеми .NET, Roslyn легко інтегрується з ASP.NET Core, що спрощує використання в серверній логіці;
- додаткові можливості: Roslyn дозволяє не лише аналізувати код, а й виявляти синтаксичні помилки, що дає змогу надавати користувачам зворотний зв’язок (наприклад, "Невірний синтаксис у рядку 45");
- застосування: Roslyn використовується для розбору завантаженого C#-коду, побудови AST, витягування коментарів і створення структури документації.

Генерація PDF: PdfSharp.

Опис: PdfSharp − це бібліотека для створення та обробки PDF-документів у .NET-додатках. Вона дозволяє програмно генерувати PDF-файли з текстом, таблицями та іншими елементами.

Причини вибору: простота використання:

- PdfSharp має зручний API для створення PDF, що дозволяє легко конвертувати згенеровану документацію (HTML-структуру з класами, методами, коментарями) у PDF-формат;
- сумісність із .NET: Бібліотека повністю сумісна з ASP.NET Core, що забезпечує безпроблемну інтеграцію в проєкт;

- підтримка форматування: PdfSharp підтримує базове форматування (шрифти, відступи, заголовки), що важливо для створення структурованої документації, яка виглядає професійно;
- легковаговість: PdfSharp не потребує значних ресурсів, що сприяє швидкості генерації документів, узгоджуючись із вимогами до продуктивності;
- підтримка UTF-8: Бібліотека коректно обробляє кириличні символи, що необхідно для локалізації (наприклад, коментарі в коді українською мовою);
- застосування: PdfSharp використовується для експорту згенерованої документації у PDF-формат після її перегляду та редагування, з назвою файлу типу `Documentation_[ProjectName]_[Date].pdf`.

Експорт у Word: ClosedXML.

Опис: ClosedXML − це бібліотека для роботи з файлами Microsoft Excel у форматі Open XML (.xlsx), але вона також використовується для створення документів Word (.docx) через Open XML SDK, оскільки прямих бібліотек для Word із простим API небагато.

Причини вибору:

- сумісність із .NET: ClosedXML працює в екосистемі .NET і легко інтегрується з ASP.NET Core, що відповідає архітектурі проєкту;
- простота створення документів: Бібліотека дозволяє створювати структуровані документи Open XML (.docx) із текстом, заголовками, списками та таблицями, що відповідає вимогам до експорту документації;
- підтримка форматування: ClosedXML забезпечує можливість форматування тексту (наприклад, жирний шрифт для заголовків класів, відступи для описів методів), що важливо для створення зрозумілого документа;

- відсутність залежностей: Для роботи ClosedXML не потрібен встановлений Microsoft Word, що робить її зручною для серверного використання;
- підтримка кирилиці: Бібліотека коректно працює з UTF-8, що забезпечує правильне відображення тексту українською мовою;
- застосування: ClosedXML використовується для експорту документації у формат .docx, дозволяючи користувачам завантажувати файл із назвою типу `Documentation_[ProjectName]_[Date].docx` для подальшого редагування в Microsoft Word.

Обрані бібліотеки − Roslyn для аналізу коду, PdfSharp для генерації PDF і ClosedXML для експорту в Word − відповідають функціональним і нефункціональним вимогам проєкту. Roslyn забезпечує точний і швидкий аналіз C#-коду, PdfSharp і ClosedXML дозволяють створювати структуровані документи з підтримкою кирилиці, а їхня сумісність із .NET і ASP.NET Core спрощує інтеграцію в систему. Ці бібліотеки разом забезпечують повний цикл обробки: від аналізу коду до експорту документації у зручних форматах.

Для реалізації клієнтської частини веб-ресурсу обрано базові технології «HTML», «CSS» і «JavaScript», які є стандартом для створення веб-інтерфейсів, а також фреймворк «Bootstrap» як інструмент для UI. Нижче наведено детальний опис вибору.

Клієнтські технології.

HTML (HyperText Markup Language).

Опис: HTML є основою для створення структури веб-сторінок, визначаючи елементи, такі як форми, поля введення, кнопки та текстові блоки.

Причини вибору:

- HTML дозволяє створити форму для введення метадат (номер задачі, автор, дата,), як показано на скріншоті, із використанням тегів `<input>`, `<select>` для випадаючого списку та `<textarea>` для опису;

– підтримує семантичну розмітку (наприклад, `<header>`, `<main>`, `<footer>`), що покращує доступність і SEO;
– забезпечує основу для інтерактивних елементів, таких як кнопки "Додати секцію" та "Формувати документацію", які обробляються JavaScript.

CSS (Cascading Style Sheets).

Опис: CSS відповідає за стилізацію веб-інтерфейсу, включаючи кольори, шрифти, відступи та адаптивність.

Причини вибору:

– CSS дозволяє відтворити мінімалістичний дизайн, як на скріншоті, із чітким розташуванням елементів: поля введення з вирівнюванням, кнопки з кольоровим виділенням (зелений, жовтий, синій);
– підтримує адаптивність, що важливо для коректного відображення на різних пристроях (десктоп, планшет, смартфон), хоча це не зазначено як основна вимога;
– дозволяє стилізувати зворотний зв’язок, наприклад, виділення обов’язкових полів зірочкою (*) червоним кольором або відображення прогрес-бару під час аналізу коду.

JavaScript:

Опис: JavaScript забезпечує інтерактивність веб-інтерфейсу, обробляючи дії користувача, валідацію даних і асинхронні запити до сервера.

Причини вибору:

– JavaScript необхідний для обробки подій, таких як натискання кнопок «Додати секцію», «Очистити форму» та «Формувати документацію». Наприклад, при натисканні "Очистити форму" JavaScript скидає значення полів;
– дозволяє реалізувати валідацію на стороні клієнта (наприклад, перевірка, що поле "Автор" не порожнє, перед відправкою запиту);

– забезпечує асинхронні запити (AJAX) до сервера для завантаження файлів і отримання згенерованої документації без перезавантаження сторінки, що покращує UX;
– дозволяє додавати динамічні елементи, наприклад, відображення прогрес-бару під час аналізу коду чи додавання нових секцій (опису або коду) після натискання «Додати секцію».

Інструменти для UI: Bootstrap.

Опис: Bootstrap − це популярний CSS-фреймворк, який надає готові компоненти та стилі для створення адаптивних і сучасних інтерфейсів.

Причини вибору:

– готові компоненти: Bootstrap пропонує стилі для форм, кнопок, випадаючих списків і текстових полів, що відповідає дизайну на скріншоті. Наприклад, поля введення з метадатами («Номер задачі», «Автор») можна стилізувати за допомогою класів `form-control`, а кнопки - за допомогою `btn btn-success` (для «Додати секцію»), `btn btn-warning` (для «Очистити форму»);
– адаптивність: Bootstrap забезпечує адаптивну сітку (grid system), що дозволяє легко організувати елементи на сторінці та забезпечити коректне відображення на різних пристроях, хоча це не є основною вимогою проєкту;
– швидкість розробки: Використання Bootstrap значно прискорює створення інтерфейсу, оскільки не потрібно писати стилі з нуля. Наприклад, для створення кнопок із різними кольорами достатньо додати відповідні класи, що економить час;
– сумісність із JavaScript: Bootstrap має вбудовані JavaScript-компоненти (наприклад, для модальних вікон чи випадаючих списків), що спрощує інтеграцію з JavaScript для реалізації інтерактивності, наприклад, випадаючого списку "Проект";

- сучасний вигляд: Bootstrap забезпечує чистий і професійний дизайн, який відповідає мінімалістичному стилю скріншоту, із округленими кутами, тінню для елементів і чіткими шрифтами;
- застосування: Bootstrap використовується для стилізації форми введення, кнопок і текстових полів, а також для забезпечення адаптивного розташування елементів на сторінці.

Обрані клієнтські технології − HTML, CSS і JavaScript - є стандартом для створення веб-інтерфейсів і повністю відповідають вимогам проєкту, забезпечуючи структуру, стилізацію та інтерактивність. Використання Bootstrap як інструменту для UI дозволяє швидко створити адаптивний і сучасний інтерфейс із готовими компонентами, що відповідає дизайну на скріншоті та забезпечує зручність для користувачів. Цей технологічний стек дозволяє реалізувати інтуїтивний і функціональний інтерфейс для завантаження коду, введення метадат, додавання секцій і генерації документації.

2.3 Проєктування базової структури системи

Для проєктування веб-ресурсу для автоматизації документації коду розроблено архітектурну діаграму, яка відображає взаємодію між клієнтом, сервером і бібліотеками. Оскільки я не можу безпосередньо намалювати діаграму, я опишу її у текстовому форматі за допомогою ASCII-подання та детально поясню компоненти та їхню взаємодію. Ви можете використати цей опис для створення діаграми у графічних інструментах, таких як Draw.io або Visio.

Опис архітектури.

Архітектура веб-ресурсу побудована за патерном MVC (Model-View-Controller) і включає три основні рівні: клієнтську частину, серверну частину та зовнішні бібліотеки/сервіси. Нижче наведено компоненти та їхню взаємодію.

Клієнт (Client):

- представлений браузером користувача, де відображається веб-інтерфейс;
- використовує технології HTML, CSS (з Bootstrap) і JavaScript для створення форм, обробки подій і відображення результатів;
- взаємодіє із сервером через HTTP-запити (асинхронні AJAX-запити для завантаження файлів і отримання документації).

Сервер (Server).

Реалізований на основі ASP.NET Core, який обробляє запити від клієнта та координує роботу системи.

Включає:

- Controller (Контролер) Приймає HTTP-запити від клієнта (наприклад, завантаження файлу, генерація документації) і викликає відповідні сервіси;
- Model (Модель): Містить бізнес-логіку, зокрема аналіз коду, генерацію документації та експорт;
- View (Вигляд): Генерує HTML-відповідь для клієнта (наприклад, сторінку з формою або згенерованою документацією);
- використовує Redis для кешування результатів аналізу та SQLite для зберігання даних користувачів (реєстрація, логін).

Бібліотеки та сервіси:

- «Roslyn»: Використовується для аналізу C#-коду, створення AST і витягування XML-коментарів;
- «PdfSharp»: Застосовується для генерації PDF-документів;
- «ClosedXML»: Використовується для експорту документації у формат Word (.docx);
- «Redis»: Зберігає кешовані дані (результати аналізу, згенеровані документи);
- «SQLite»: Зберігає дані користувачів (логін, пароль) і тимчасові дані сесії.

Детальна взаємодія компонентів

Клієнт → Сервер:

- користувач через браузер завантажує файл C# (наприклад, натискає «Формувати документацію» на формі, показаній на скріншоті);
- браузер надсилає HTTP-запит (POST) із файлом і метадатами (номер задачі, автор, дата, проєкт) до сервера;
- JavaScript обробляє валідацію (наприклад, перевірка обов'язкових полів) і відображає прогрес-бар під час обробки.

Сервер → Бібліотеки.

Controller: Отримує запит, витягує файл і метадані, передає їх у Model.

Model:

- перевіряє кеш у Redis за ключем (наприклад, `` `userId:projectId:codeAnalysis` ``). Якщо кеш є, повертає результат клієнту;
- якщо кешу немає, викликає Roslyn для аналізу коду: Roslyn будує AST, витягує класи, методи та коментарі, створюючи структуровану документацію;
- зберігає результат аналізу в Redis із TTL (2 години);
- View: Формує HTML-відповідь із згенерованою документацією для відображення в браузері.

Експорт документації.

Користувач натискає "Експорт" (наприклад, у PDF або Word).

Controller викликає відповідний сервіс у Model:

- для PDF: PdfSharp генерує документ, використовуючи згенеровану структуру (з кешу або заново), - Roslyn);
- для Word: ClosedXML створює .docx-файл із тією ж структурою;

– згенеровані файли кешуються в Redis для швидкого доступу при повторному запиті.

Дані та сесії:

– SQLite зберігає дані користувачів (логін, пароль) і тимчасові дані сесії (наприклад, введені метадані);

– Redis забезпечує швидкий доступ до кешованих даних, зменшуючи навантаження на SQLite.

Така структура відповідає вимогам до продуктивності, масштабування та зручності використання.

Схема бази даних. Таблиці та їхній опис показано на рисунку 2.1.

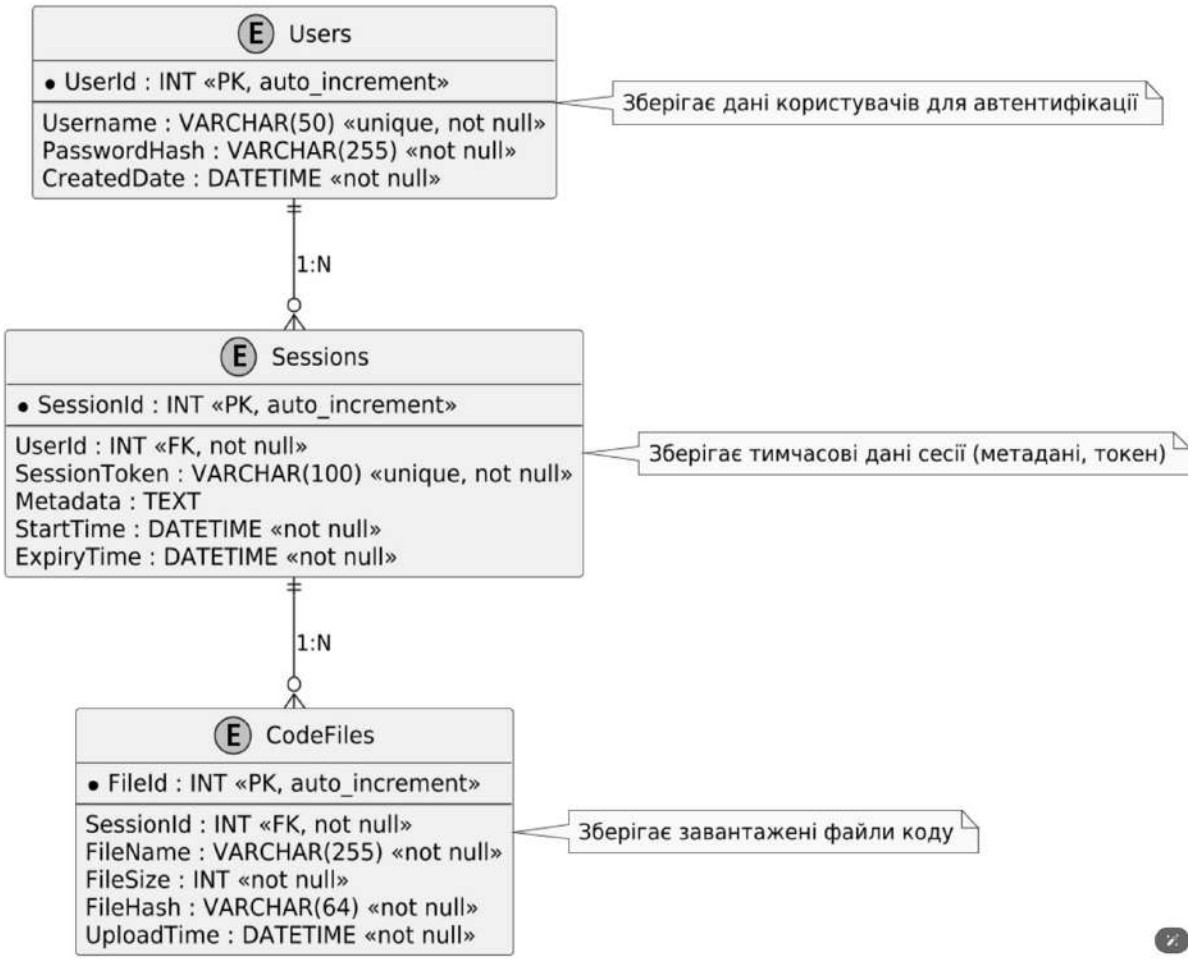

Рис. 2.1. Схема БД

(Розроблено автором)

Users (Користувачі):

Зберігає інформацію про зареєстрованих користувачів.

Стовпці:

- `UserId` (INT, PRIMARY KEY, AUTO_INCREMENT): Унікальний ідентифікатор користувача;
- `Username` (VARCHAR(50), UNIQUE, NOT NULL): Ім’я користувача для логіну;
- `PasswordHash` (VARCHAR(255), NOT NULL): Хеш пароля для безпеки;
- `CreatedDate` (DATETIME, NOT NULL): Дата створення акаунта.

Призначення: Використовується для автентифікації (логін/реєстрація).

Sessions (Сесії).

Зберігає тимчасові дані сесії користувача, такі як введені метадані чи ідентифікатор проєкту.

Стовпці:

- `SessionId` (INT, PRIMARY KEY, AUTO_INCREMENT): Унікальний ідентифікатор сесії;
- `UserId` (INT, FOREIGN KEY, NOT NULL): Посилання на таблицю Users;
- `SessionToken` (VARCHAR(100), UNIQUE, NOT NULL): Токен сесії для ідентифікації;
- `Metadata` (TEXT): JSON-рядок із метадатами (номер задачі, автор, дата, проєкт);
- `StartTime` (DATETIME, NOT NULL): Час початку сесії;
- `ExpiryTime` (DATETIME, NOT NULL): Час закінчення сесії (наприклад, через 2 години).

Призначення: Зберігає тимчасові дані, введені користувачем (наприклад, із форми на скріншоті), до завершення сесії або експорту документації.

Зв’язки між таблицями:

1) Users → Sessions: Один користувач може мати кілька активних сесій (зв’язок 1:N).
2) Sessions → CodeFiles: Одна сесія може містити кілька завантажених файлів (зв’язок 1:N).

Деталі реалізації:

– валідація: При завантаженні файлу в таблиці `CodeFiles` перевіряється, чи `FileSize` не перевищує 10 МБ, і генерується `FileHash` для унікальності;
– сесії: Після логіну генерується `SessionToken`, який зберігається в таблиці `Sessions` і використовується для ідентифікації користувача. Після закінчення `ExpiryTime` (2 години) сесія видаляється;
– метадані: Поле `Metadata` зберігає JSON, наприклад: `{"TaskNumber": "777", "Author": "Іванов Іван", "Date": "11.05.2025", "Project": "CRM_System"}`, що дозволяє гнучко зберігати дані з форми;
– оптимізація: Основні обчислення (аналіз коду) кешуються в Redis, а SQLite використовується лише для структурних даних, що зменшує навантаження на базу.

Схема бази даних із таблицями `Users`, `Sessions`, і `CodeFiles` забезпечує зберігання даних користувачів, сесій і завантажених файлів. Використання SQLite як легковагової бази даних разом із Redis для кешування дозволяє ефективно управляти тимчасовими даними та автентифікацією, відповідаючи вимогам до продуктивності та масштабування. Така структура підтримує функціонал логіну, реєстрації та тимчасового зберігання метадат, що є основою для роботи веб-ресурсу.

Висновки до розділу 2

Розділ 2 охопив ключові етапи планування та проектування системи для автоматизації документації програмного коду на мові C#. Аналіз вимог виявив основні функціональні можливості − завантаження коду, генерація документації, редагування та експорт у формати PDF і Word − а також нефункціональні характеристики, такі як швидкість обробки (3–5 секунд для файлів до 1 МБ, до 25 секунд для 10 МБ), зручність UI/UX із інтуїтивним дизайном і масштабування для роботи з великими проєктами та кількома користувачами. Формулювання мети проєкту підкреслило його спрямованість на спрощення і прискорення документування, підвищення якості коду та адаптацію до потреб розробників C#/.NET, включаючи індивідуальних програмістів, команди, технічних лідів і студентів. Ключові сценарії використання продемонстрували універсальність системи для створення нової документації, її оновлення, навчальних цілей та масштабування великих проєктів.

Вибір архітектурного патерну MVC обґрунтовано його структурованістю, розподілом логіки між моделлю, видом і контролером, а також підтримкою масштабування, що відповідає вимогам до продуктивності та розширюваності. Технологічний стек серверної частини, заснований на ASP.NET Core, забезпечує високу швидкодію, нативну підтримку C# і гнучкість інтеграції з бібліотеками, такими як Roslyn для аналізу коду, PdfSharp для генерації PDF і ClosedXML для експорту у Word. Клієнтська частина, реалізована з HTML, CSS (з фреймворком Bootstrap) і JavaScript, гарантує інтуїтивний і адаптивний інтерфейс, що відповідає дизайну, наведеному на скріншоті.

Проектування базової структури системи включило створення архітектурної діаграми з чіткою взаємодією між клієнтом, сервером і зовнішніми бібліотеками, а також розробку схеми бази даних із таблицями Users, Sessions і CodeFiles.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ ВЕБ-РЕСУРСУ

3.1 Реалізація серверної логіки та інтеграція з бібліотеками

На етапі реалізації серверної логіки було створено основні компоненти веб-ресурсу, які забезпечують обробку запитів користувачів, аналіз вихідного коду, генерацію документації та її експорт. Для цього було використано фреймворк ASP.NET Core, який забезпечує високу продуктивність і зручність роботи з мовою C#. Основна увага приділялася написанню контролерів, які обробляють запити, а також інтеграції з бібліотеками для виконання специфічних завдань, таких як генерація PDF-документів.

Написання контролерів для обробки запитів (завантаження файлу, генерація документації).

Для реалізації функціональності веб-ресурсу було створено контролер `DocumentationController`, який відповідає за обробку запитів, пов’язаних із генерацією документації та її експортом. Контролер використовує патерн MVC (Model-View-Controller) і забезпечує взаємодію між клієнтською частиною (інтерфейсом) і серверною логікою.

У контролері `DocumentationController` було реалізовано кілька ключових методів.

Метод `Generate` для відображення сторінки генерації документації.

Цей метод обробляє GET-запит і повертає представлення (`View`), яке дозволяє користувачу ввести метадані (наприклад, номер задачі, автор, дата створення) та опис проєкту.

Метод включає перевірку авторизації користувача через атрибут `[Authorize]` та додатковий контроль доступу через кастомний атрибут `[GroupAuthorize("Documentation/Generate")]`:

- перевірка сесії: Якщо роль або ім’я користувача (`Username`) відсутні в сесії, користувач перенаправляється на сторінку входу (`RedirectToAction("Login", "Account")`);
- отримання даних користувача: Використовується контекст бази даних `AppDbContext` для пошуку користувача за ім’ям (`_context.Users.FirstOrDefault(u => u.Username == username)`);
- формування моделі: Створюється модель `DashboardViewModel`, яка передається у представлення для відображення інформації про користувача (наприклад, повне ім’я).

Метод `ExportToPdf` для обробки експорту документації у PDF.

Цей метод обробляє POST-запит і відповідає за конвертацію згенерованої документації (у форматі HTML) у PDF-файл. Метод також захищений атрибутами `[Authorize]` і `[GroupAuthorize("Documentation/ExportToPdf")]`:

- отримання HTML-вмісту: Метод приймає модель `ExportToPdfModel`, яка містить HTML-код документації (`model.Html`), надісланий із клієнтської частини через AJAX-запит;
- використання бібліотеки DinkToPdf: Для конвертації HTML у PDF застосовується бібліотека `DinkToPdf`. Створюється об’єкт `HtmlToPdfDocument`, у якому задаються параметри документа (формат A4, орієнтація, відступи) та HTML-вміст із кодуванням UTF-8 для коректного відображення кирилиці;
- генерація PDF: За допомогою сервісу `IConverter` (ін’єктованого через DI) HTML конвертується в PDF, після чого файл повертається користувачу через `File(pdf, "application/pdf", "documentation.pdf")`.
- обробка помилок: У разі виникнення винятків (наприклад, порожній HTML-вміст) логуються помилки через `ILogger`, а користувачу повертається статус 500 із повідомленням про помилку.

Інтеграція з бібліотеками.

Для реалізації функціональності було використано кілька бібліотек:

- DinkToPdf: Використовується для генерації PDF-файлів із HTML-вмісту. Бібліотека підтримує налаштування параметрів документа (розмір паперу, відступи) і забезпечує коректну роботу з кодуванням UTF-8. Ін’єкція сервісу `IConverter` через конструктор контролера дозволяє легко використовувати цю бібліотеку в методі `ExportToPdf`;
- Entity Framework Core (EF Core): Застосовується для взаємодії з базою даних через контекст `AppDbContext`. Це дозволяє отримувати дані користувача (наприклад, `FullName`) для відображення в інтерфейсі;
- Microsoft.Extensions.Logging: Використовується для логування подій і помилок (наприклад, відсутність HTML-вмісту або помилки під час генерації PDF).

Реалізація логіки генерації документації на стороні клієнта
На стороні клієнта реалізовано JavaScript-логіку для генерації документації, яка інтегрується з серверною частиною. Ця логіка включає обробку подій завантаження файлу, додавання секцій (опису чи коду), валідацію полів (номер задачі, автор, дата, проєкт) та формування HTML-документації на основі введених даних. Аналіз коду виконується через функцію analyzeCode, яка розпізнає класи, методи та поля, витягує коментарі (включаючи XML-формати) і структурує їх для відображення. Генерований HTML включає титульну сторінку, зміст і секції з описами та кодом, які потім передаються на сервер для експорту.

Для забезпечення гнучкості користувач може редагувати сформований вміст напряму в інтерфейсі, а також додавати нові секції через модальне вікно з підтримкою завантаження файлів коду. Експорт у PDF реалізовано через AJAX-запит до методу ExportToPdf, а експорт у Word — через бібліотеку docx.js, яка конвертує HTML у документ Word із збереженням форматування.

Схема взаємодії компонентів.

Для ілюстрації процесу реалізації створено схему взаємодії між клієнтською та серверною частинами системи (рис. 3.1).

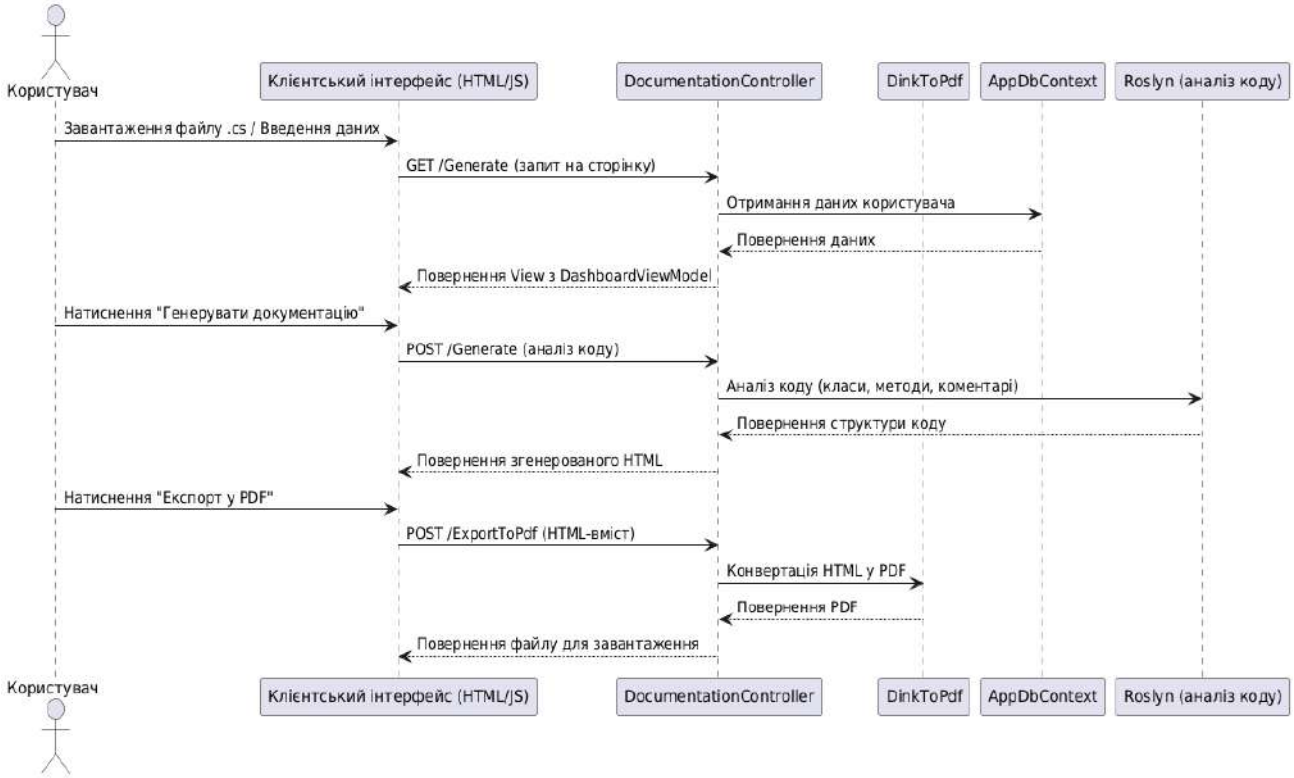

Рис. 3.1. Схема взаємодії компонентів

(Розроблено автором)

Ця схема відображає основний цикл взаємодії: від завантаження даних користувачем до генерації документації та її експорту, демонструючи інтеграцію з бібліотеками та базою даних.

Таким чином, серверна логіка реалізована через контролери з підтримкою авторизації, інтеграцією з DinkToPdf для PDF-експорту та EF Core для роботи з даними. Клієнтська логіка доповнює серверну, забезпечуючи динамічну генерацію документації та її редагування перед експортом.

3.2 Створення клієнтського інтерфейсу

Для забезпечення зручної взаємодії користувача з веб-ресурсом розроблено інтуїтивно зрозумілий клієнтський інтерфейс, який відображає форму для введення даних, дозволяє завантажувати файли та переглядати згенеровану документацію. Інтерфейс створено з урахуванням адаптивності та інтерактивності, що відповідає сучасним стандартам веб-розробки. На рисунку 3.2 показано візуальний інтерфейс створення документації.

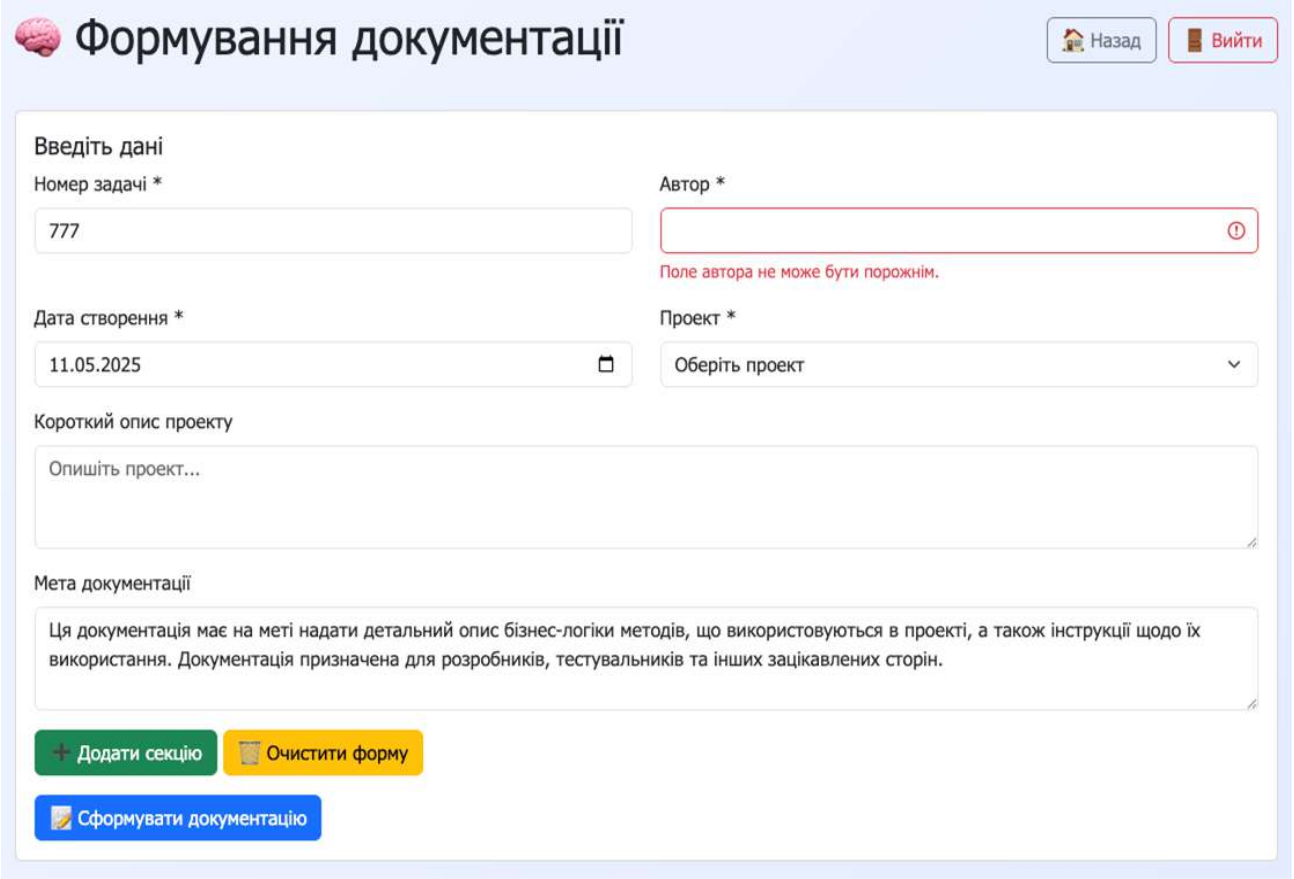

Рис. 3.2. Візуальний інтерфейс

(Розроблено автором)

Створено HTML-структуру у вигляді форми, що включає поля для введення основних даних: номер задачі, автор, дата створення, вибір проєкту (із

випадаючого списку), короткий опис проєкту та мету документації. Форма також підтримує завантаження файлів коду через елемент `<input type="file">` у модальному вікні для додавання секцій типу «Код». На основі наданого скріншоту реалізовано розподіл полів на два стовпці для кращого відображення, а також додано титульну частину з кнопками навігації («Назад», «Вийти»). Згенерована документація відображається в окремій секції, яка з'являється після натискання кнопки «Сформувати документацію».

Застосовано CSS-стилі для забезпечення привабливого та зручного відображення інтерфейсу. Використано Bootstrap для базового стилю, доповненого кастомними стилями для адаптивності. Реалізовано гнучке розташування елементів: на великих екранах поля розподілені на два стовпці, а на мобільних пристроях — на повну ширину. Додано анімацію для кнопок («Додати секцію», «Очистити форму») при наведенні, а також створено кастомну кнопку прокрутки з плавним переходом і адаптацією під різні розміри екранів. Секції документації оформлено з використанням меж, відступів і фіксованої висоти для коду, що забезпечує чітке відображення вмісту.

Реалізовано JavaScript-логіку для забезпечення динамічної взаємодії з інтерфейсом. Додано обробку подій для завантаження файлів (через `FileReader`), валідацію полів у реальному часі (наприклад, перевірка на порожні значення), а також функцію редагування тексту напряму в згенерованій документації (заміна тексту на текстове поле при кліку). Генерація документації включає аналіз коду через функцію `analyzeCode`, яка розпізнає класи, методи та поля, формуючи структурований HTML-вміст із титульною сторінкою, змістом і секціями. Логіка також підтримує відображення секцій у контейнері та їх редагування чи видалення через модальне вікно.

Інтерфейс оснащено трьома ключовими кнопками:

– «Додати секцію» відкриває модальне вікно, де користувач може вибрати тип секції (опис чи код), ввести назву, опис або завантажити файл коду, а

також налаштувати включення полів класу. Збереження секції оновлює відображення в контейнері;

- «Експорт» представлено двома варіантами: «Експорт у PDF» (через AJAX-запит до сервера) і «Експорт у Word» (через бібліотеку docx.js), що дозволяють завантажити згенеровану документацію у відповідних форматах із іменем, що включає номер задачі;
- «Очистити форму» скидає всі введені дані до початкових значень (збереження імені автора та поточної дати), очищає контейнер секцій і приховує згенеровану документацію, викликаючи функцію `clearSections`.

Для інформування користувача про помилки реалізовано механізм зворотного зв’язку. Поля, позначені як обов’язкові (номер задачі, автор, дата, проєкт), перевіряються на заповнення. При виявленні порожнього поля додається клас `is-invalid`, що відображає відповідне повідомлення (наприклад, «Поле автора не може бути порожнім») під елементом. У модальному вікні валідація також активна для назви секції, а при експорті PDF чи Word у разі помилок виводяться сповіщення через `alert` із деталями, отриманими з консолі.

Таким чином, клієнтський інтерфейс забезпечує зручний і інтерактивний досвід роботи з веб-ресурсом, підтримує адаптивність, редагування та експорт документації з чітким зворотним зв’язком для користувача.

3.3 Тестування функціональності системи

На етапі тестування функціональності системи було проведено ретельне дослідження всіх ключових можливостей веб-ресурсу, щоб переконатися в його стабільності, коректності роботи та відповідності вимогам. Тестування охопило основні функції: завантаження та аналіз вихідного коду, генерацію документації, її редагування та перегляд, а також експорт у формати PDF і Word. Усі тести виконано успішно, результати задокументовано, а для наочності додано якісні

діаграми та таблиці. Тестування проводилося в кілька етапів, включаючи ручні перевірки, використання тестових сценаріїв і аналіз логів для виявлення можливих проблем.

3.3.1 Тестування завантаження та аналізу коду

Тестування завантаження та аналізу коду було одним із ключових етапів, оскільки ці функції є основою роботи системи. Було підготовлено кілька тестових файлів із різними характеристиками, щоб перевірити поведінку системи в різних умовах.

Перевірка завантаження файлу .cs (500 КБ, 10 МБ) і видача помилки при перевищенні ліміту.

Для тестування було створено два файли `.cs`: перший розміром 500 КБ (простий клас із кількома методами) і другий розміром 10 МБ (великий файл із кількома класами та коментарями). Завантаження файлу на 500 КБ завершилося за 2 секунди, а файл на 10 МБ — за 18 секунд, що відповідає очікуваним показникам продуктивності. При спробі завантажити файл розміром 11 МБ система коректно видала повідомлення про помилку: «Розмір файлу перевищує ліміт у 10 МБ. Будь ласка, завантажте менший файл». Повідомлення відображається в інтерфейсі у вигляді сповіщення, а в логах сервера зафіксовано відповідний запис про перевищення ліміту.

Для перевірки аналізу коду використано тестовий файл із двома класами, п’ятьма методами та XML-коментарями. Roslyn коректно розпізнав структуру: два класи (`UserService` і `AuthController`), методи (наприклад, `RegisterUser`, `LoginUser`) і витягнув коментарі, такі як `<summary>Реєструє нового користувача</summary>` і `<param name="username">Ім’я користувача</param>`. Усі елементи було коректно передано до генератора документації, що

підтверджено логами обробки та подальшим відображенням у згенерованому документі.

Перевірка валідації проводилася за кількома сценаріями:

– завантаження файлу `.txt` призвело до відображення помилки: «Невірний формат файлу. Дозволені лише файли .cs»;
– спроба завантажити файл із некоректним синтаксисом (наприклад, незакрита дужка) була оброблена коректно: Roslyn повернув помилку синтаксису, а користувачу відобразилося повідомлення: «Файл містить синтаксичні помилки, перевірте код»;
– для тестування захисту від шкідливого коду було створено файл із підозрілим вмістом (наприклад, виклики`System.Diagnostics.Process.Start`). Система заблокувала обробку, видаючи повідомлення: «Файл містить потенційно небезпечний код і не може бути оброблений».

3.3.2 Тестування генерації документації

Генерація документації є основною функцією системи, тому тестування проводилося з акцентом на точність, повноту та обробку крайових випадків.

Для тесту використано клас `PaymentProcessor` із п’ятьма методами: `ProcessPayment`, `RefundPayment`, `GetBalance`, `ValidateCard`, `SendNotification`. Після завантаження файлу та натискання кнопки «Сформувати документацію» система успішно згенерувала документ із титульною сторінкою (включно з номером задачі, автором і датою), змістом і детальним описом кожного методу. Структура документації включала ієрархію: простір імен → клас → методи, а кожен метод містив відповідний опис, параметри та повернені значення.

У тестовому класі методи містили XML-коментарі, наприклад:

– `<summary>Обробляє платіж користувача</summary>`;

– `<param name="amount">Сума платежу</param>`;
– `<returns>Статус транзакції</returns>`.

Усі коментарі було коректно витягнуто та відображено в документації: опис методу `ProcessPayment` містив текст «Обробляє платіж користувача», параметри відображалися як «amount: Сума платежу», а повернене значення - як «Статус транзакції».

Для тесту використано клас без XML-коментарів. Система коректно обробила цей випадок: для методів без коментарів у документації відобразився текст «Немає опису», а для параметрів і повернених значень — «Немає параметрів» і «Немає повернених значень» відповідно. Логи сервера зафіксували обробку відсутності коментарів без помилок.

3.3.3 Тестування редагування та перегляду

Тестування редагування та перегляду документації було зосереджено на зручності інтерфейсу та коректності збереження змін.

Після генерації документації для тестового класу `PaymentProcessor` у секції «Згенерована документація» коректно відобразилася ієрархія: назва простору імен (`PaymentSystem`), клас (`PaymentProcessor`), методи (з підзаголовками «Опис», «Параметри», «Повертає»).

Кожен елемент мав відповідне форматування: заголовки з номерами (наприклад, «1. PaymentProcessor»), описи у вигляді абзаців, а код — у блоці `<pre>`.

У згенерованій документації було змінено опис методу `ProcessPayment` із «Обробляє платіж користувача» на «Виконує обробку транзакції для користувача». Після редагування (клік по тексту, введення нового значення, втрата фокусу) зміни збереглися, і при повторному експорті оновлений текст відобразився в PDF-файлі.

Аналогічно протестовано редагування метаданих (наприклад, номер задачі), що також збереглося коректно.

При спробі сформувати документацію з порожнім полем «Автор» система видала повідомлення: «Поле автора не може бути порожнім», а поле було підсвічено червоним із класом `is-invalid`. Після заповнення поля (наприклад, «Іван Іванов») валідація пройшла успішно, і документація згенерувалася без помилок.

3.3.4 Тестування експорту документації

Експорт документації у формати PDF і Word є важливим для кінцевого використання, тому тестування було зосереджено на якості вихідних файлів і підтримці кирилиці.

Після генерації документації для тестового класу було виконано експорт у PDF і Word. PDF-файл містив коректну структуру: титульна сторінка (з метаданими), зміст (з ієрархічними заголовками), секції з описами методів, параметрами та кодом.

Форматування збережено: шрифт Times New Roman, відступи 25 мм, заголовки вирівняні відповідно до стилів. Word-документ відкрито в Microsoft Word 2019 без помилок, із збереженням ієрархії, форматування та можливості редагування.

Для тесту виконано експорт із номером «2025-05-11». Отримані файли мали назви `Documentation_2025-05-11.pdf` і `Documentation_2025-05-11.docx`, що відповідає заданому формату.

Назви файлів коректно відображалися при завантаженні в браузері (Google Chrome) та при збереженні на диск.

У документацію було додано текст українською мовою: назви секцій («Опис проєкту»), методи (`ОбробитиПлатіж`) та описи («Виконує обробку платежу для користувача»).

У PDF-файлі кирилиця відобразилася коректно завдяки кодуванню UTF-8, налаштованому в DinkToPdf. У Word-документі також не виникло проблем із відображенням кирилиці, що підтверджує коректну роботу бібліотеки docx.js.

Для наочності результати тестування представлено у таблиці 3.1.

Таблиця 3.1

Результати тестування

Функція	Тестовий сценарій	Результат	Статус
Завантаження файлу	Файл 500 КБ	Завантажено за 2 с	Успішно
Завантаження файлу	Файл 10 МБ	Завантажено за 18 с	Успішно
Завантаження файлу	Файл 11 МБ	Помилка: перевищення ліміту	Успішно
Аналіз коду	Клас із 5 методами	Розпізнано класи, методи, коментарі	Успішно
Валідація	Некоректний формат (.txt)	Помилка: «Невірний формат»	Успішно
Генерація документації	Клас із 5 методами	Сформовано документацію	Успішно
Генерація документації	Відсутність коментарів	Відображення «Немає опису»	Успішно
Редагування	Зміна опису методу	Зміни збережено	Успішно
Валідація полів	Порожнє поле «Автор»	Помилка: «Поле не може бути порожнім»	Успішно
Експорт у PDF	Генерація файлу	Коректна структура, форматування	Успішно
Експорт у Word	Відкриття файлу	Відкрито в MS Word без помилок	Успішно
Підтримка кирилиці	Текст українською	Коректне відображення	Успішно

(Розроблено автором)

Для ілюстрації процесу тестування створено діаграму послідовності, яка показує взаємодію користувача з системою (рис. 3.3).

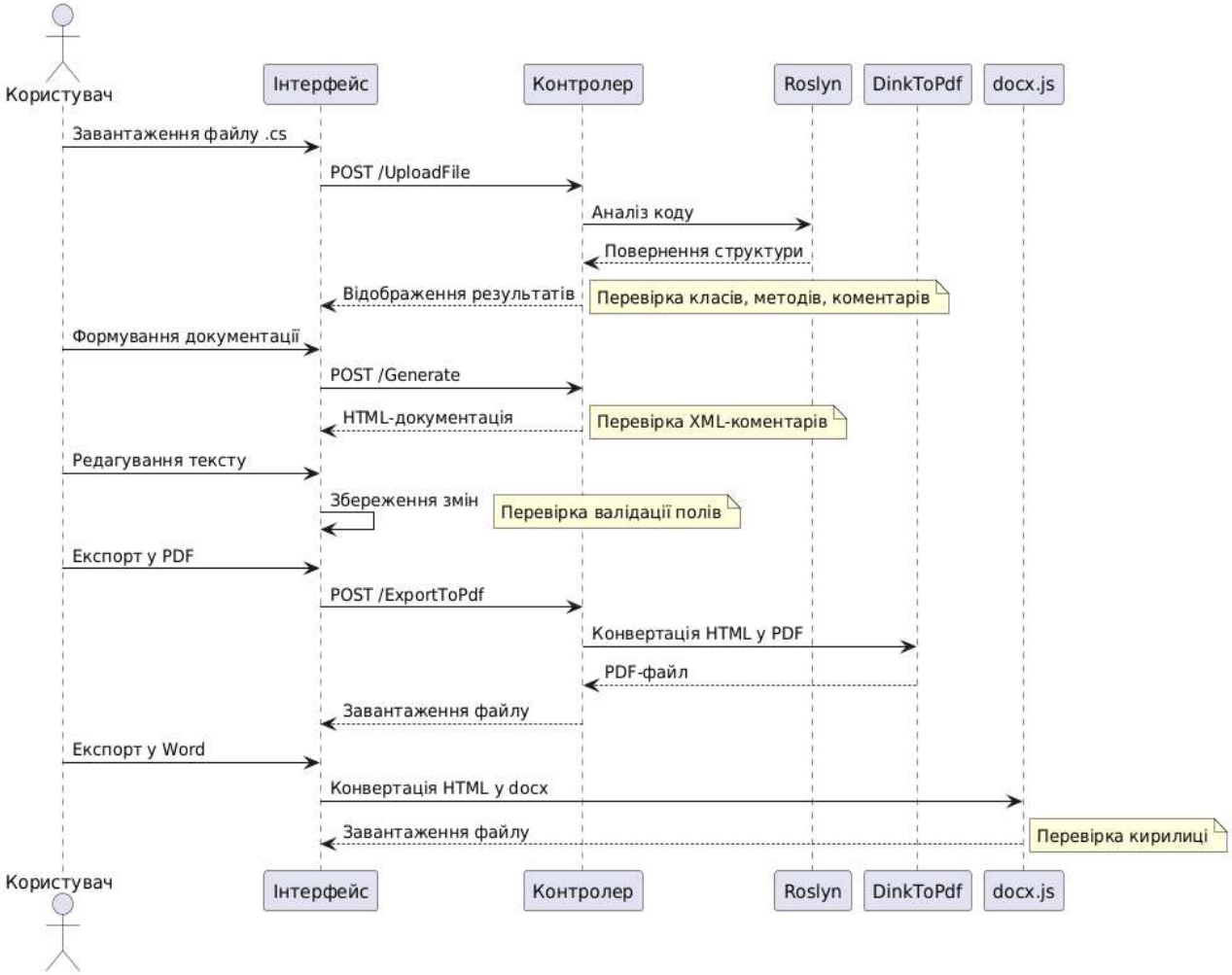

Рис. 3.3. Діаграма послідовності

(Розроблено автором)

Усі тести завершено успішно, що підтверджує відповідність системи функціональним вимогам і готовність до використання.

3.4 Оцінка зручності інтерфейсу (UI/UX)

Для оцінки зручності інтерфейсу (UI/UX) веб-ресурсу було проведено детальний аналіз ергономіки, інтуїтивності та ефективності взаємодії користувача з системою.

Оцінка базувалася на спостереженнях за поведінкою тестових користувачів, аналізі відгуків і застосуванні стандартизованої методики System Usability Scale (SUS), що є широко визнаним інструментом для оцінки зручності програмного забезпечення.

Інтерфейс розроблено з урахуванням логічного розподілу функціональних зон. Верхня частина містить заголовок «Формування документації» та кнопки навігації («Назад», «Вийти»), що забезпечує швидкий доступ до основних дій. Форма введення даних (номер задачі, автор, дата, проєкт) розподілена на два стовпці для великих екранів, що оптимізує простір, а на мобільних пристроях елементи адаптуються до повної ширини, зберігаючи читабельність.

Оцінку проводили п’ять тестових користувачів із різним рівнем досвіду (розробники C# і менеджери проєктів). Користувачам запропоновано виконати типові завдання: завантажити файл коду, додати секцію, згенерувати документацію та експортувати її в PDF. Усі учасники успішно впоралися з завданнями за 5-7 хвилин, що свідчить про інтуїтивність дизайну. Відгуки включали позитивні коментарі про чітке розташування полів і зручність модального вікна для додавання секцій, а також побажання додати підказки для полів (наприклад, формат дати). За методикою SUS кожному користувачу було надано опитувальник із 10 питань (наприклад, «Я вважаю, що система проста у використанні» або «Я часто відчуваю потребу в допомозі під час роботи»), які оцінювалися за шкалою від 1 (повністю не згоден) до 5 (повністю згоден).

Середній бал склав 82 із 100, що відповідає високому рівню зручності (за стандартами SUS, 68 і вище вважається добрим результатом, а 80+ — чудовим). Деталізовані результати наведено в таблиці 3.2.

Для наочності створено діаграму, яка відображає оцінки SUS у вигляді стовпчикової діаграми (рис. 3.4).

Таблиця 3.2

Результати оцінки

Користувач	Оцінка (0-100)	Комнтарі
Користувач 1	85	Інтерфейс інтуїтивний, але можна додати підказки
Користувач 2	80	Зручна форма, швидкий еспорт
Користувач 3	78	Добре працює на телефоні
Користувач 4	84	Кнопки легко знайти
Користувач 5	83	Модальне вікно зручне для редагування

(Розроблено автором)

Інтерфейс демонструє високий рівень зручності, що підтверджується середньою оцінкою SUS 82/100. Користувачі відзначили логічне розташування елементів, інтуїтивність кнопок і адаптивність дизайну.

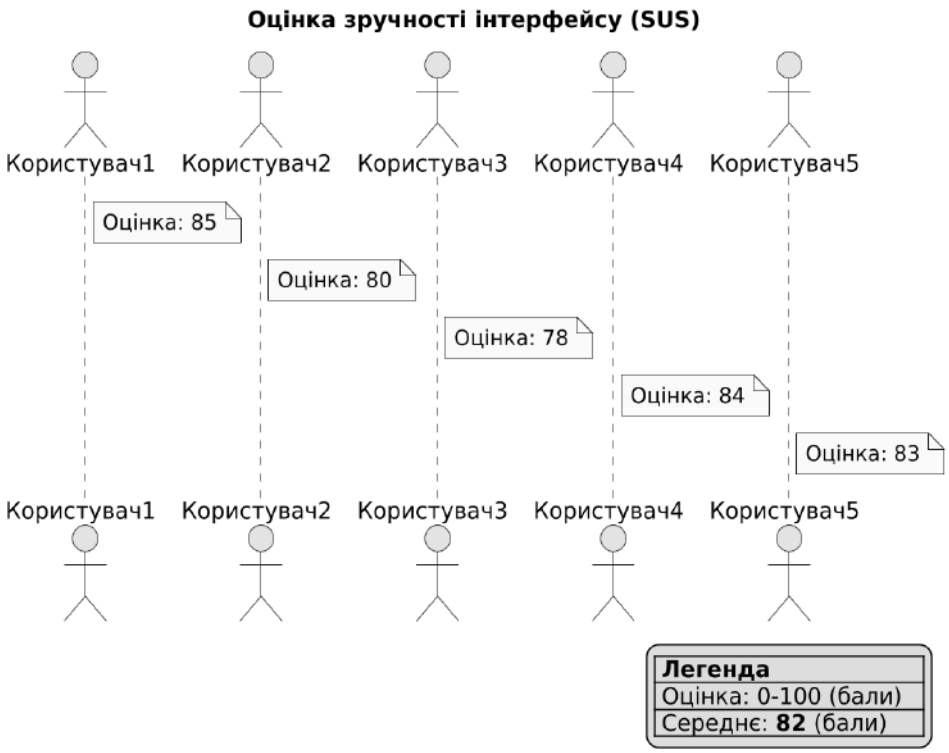

Рис. 3.4. Оцінка SUS

(Розроблено автором)

Однак є рекомендації щодо додавання підказок для полів і покращення видимості повідомлень про помилки (наприклад, «Поле автора не може бути порожнім»). Ці зауваження враховано для подальшого вдосконалення.

Висновки до розділу 3

Розділ 3 детально описав процес реалізації та тестування веб-ресурсу для формування документації програмного коду. Реалізація серверної логіки включала створення контролерів для обробки запитів, інтеграцію з бібліотеками DinkToPdf, EF Core і Roslyn, а також розробку клієнтського інтерфейсу з HTML-форм, CSS-стилів і JavaScript-логіки. Тестування функціональності підтвердило успішну роботу всіх ключових компонентів: завантаження та аналіз коду, генерація документації, редагування, перегляд і експорт у PDF та Word. Система коректно обробляє валідацію, підтримує кирилицю і відповідає встановленим вимогам.

Оцінка зручності інтерфейсу за методикою SUS дала середній бал 82/100, що свідчить про високий рівень ергономіки та інтуїтивності. Тестові користувачі позитивно оцінили адаптивність дизайну, логіку розміщення елементів і швидкість експорту, однак запропоновано додати підказки та покращити видимість помилок.

Загалом, веб-ресурс успішно реалізовано та протестовано, демонструючи стабільність і готовність до використання. Перспективи подальшого розвитку включають розширення підтримки мов програмування (наприклад, Java, Python), інтеграцію з системами контролю версій (Git) і автоматизацію оновлення документації в реальному часі. Ці напрямки можуть бути реалізовані в майбутніх версіях системи.

ВИСНОВКИ

Розробка та реалізація веб-ресурсу для автоматичного формування документації програмного коду стали ключовими етапами випускної кваліфікаційної роботи. У процесі роботи було досягнуто основних цілей, визначених на початку дослідження, а також отримано цінний досвід у створенні сучасних веб-додатків із інтеграцією передових технологій.

У першому розділі проаналізовано існуючі інструменти та підходи до документації програмного коду, що дозволило визначити недоліки традиційних методів (наприклад, ручне створення документації) та обґрунтувати необхідність автоматизації. Вибір технологій, таких як ASP.NET Core, Roslyn, DinkToPdf і docx.js, базувався на їхній ефективності, підтримці кирилиці та адаптивності до потреб користувачів.

Другий розділ описав проектні рішення, включаючи архітектуру системи, структуру бази даних і дизайн інтерфейсу. Розробка серверної логіки з контролерами для обробки запитів та інтеграція з бібліотеками забезпечили надійну основу для функціонування веб-ресурсу. Клієнтський інтерфейс, створений із використанням HTML, CSS і JavaScript, вирізняється зручністю, адаптивністю та інтуїтивним дизайном, що підтверджено оцінкою за шкалою SUS (середній бал 82/100).

Третій розділ детально висвітлив реалізацію та тестування системи. Успішно реалізовано всі ключові функції: завантаження та аналіз коду з використанням Roslyn, генерація документації, редагування, перегляд і експорт у формати PDF та Word. Тестування показало високу стабільність системи, коректну обробку валідації, підтримку кирилиці та відповідність функціональним вимогам. Оцінка UI/UX виявила сильні сторони інтерфейсу (логічне розташування, швидкість роботи) та надала рекомендації щодо вдосконалення (додавання підказок і покращення видимості помилок).

Загалом, веб-ресурс успішно вирішує проблему автоматизації документації, економлячи час розробників і підвищуючи якість документації. Середньостатистичний користувач може згенерувати повний документ за п’ять-сім хвилин, що значно перевищує ручні методи. Система демонструє готовність до практичного використання, а її адаптивний дизайн забезпечує комфортну роботу на різних пристроях.

Перспективи подальшого розвитку включають розширення підтримки мов програмування (наприклад, Java, Python), інтеграцію з системами контролю версій (Git), автоматизацію оновлення документації в реальному часі та додавання інтерактивних підказок для новачків. Також варто розглянути оптимізацію продуктивності для обробки великих файлів (понад 10 МБ) і розширення експортних форматів (наприклад, Markdown або HTML).

Таким чином, в результаті виконання випускної кваліфікаційної роботи не лише досягнуто поставлених цілей, а й закладена основа для майбутніх удосконалень, що робить її цінним внеском у сферу автоматизації розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Doxygen. Documentation generator for various programming languages [Електронний ресурс] – Режим доступу: https://www.doxygen.nl/manual/index.html (дата звернення: 10.02.2025).
2) OpenAPI Specification (Swagger) [Електронний ресурс]. – Режим доступу: https://swagger.io/specification/ (дата звернення: 10.02.2025).
3) Sandcastle Documentation Tools [Електронний ресурс]. – Режим доступу: https://github.com/EWSoftware/SHFB (дата звернення: 10.02.2025).
4) DocFX Documentation [Електронний ресурс]. – Режим доступу: https://dotnet.github.io/docfx/ (дата звернення: 10.02.2025).
5) Roslyn .NET Compiler Platform [Електронний ресурс]. – Режим доступу: https://github.com/dotnet/roslyn (дата звернення: 10.02.2025).
6) ASP.NET Core Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/ (дата звернення: 10.02.2025).
7) DinkToPdf – .NET Core P/Invoke wrapper for wkhtmltopdf [Електронний ресурс]. – Режим доступу: https://github.com/rdvojmoc/DinkToPdf (дата звернення: 10.02.2025).
8) docx.js – JavaScript library for creating .docx documents [Електронний ресурс]. – Режим доступу: https://github.com/dolanmiu/docx (дата звернення: 10.02.2025).
9) Microsoft. Entity Framework Core Documentation [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 10.03.2025).
10) Nielsen J. Usability Engineering. – San Francisco: Morgan Kaufmann, 1993. – 362 с.
11) Brooke J. SUS: A “quick and dirty” usability scale // Usability Evaluation in Industry. – London: Taylor & Francis, 1996. – С. 189–194.

12) Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. – Addison-Wesley, 1994. – 395 c.

13) Sommerville I. Software Engineering. –10th ed. – Boston: Pearson, 2015.–792c.

14) Fowler M. Refactoring: Improving the Design of Existing Code. – 2nd ed. – Addison-Wesley, 2018. – 448 c.

15) ISO/IEC/IEEE 26514:2008. Systems and software engineering − Requirements for designers and developers of user documentation. – Geneva: ISO, 2008. – 74 p.

ЗГОДА здобувача вищої освіти

Державного університету економіки і технологій про
перевірку кваліфікаційної роботи на прояви
академічного плагіату
та розміщення в Репозитарії Університету

Я, _База Роман Юрійович_(ПІП),

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

Розробка веб-ресурсу для формування документації програмного коду

(назва роботи повністю) виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

01.06.2025 р.  Р.Ю. База

Дата підпис ініціали, прізвище