ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Web-сервіс для автоматичної генерації звітів та документів за шаблоном»

на здобуття освітнього ступеня бакалавра зі спеціальності 121 Інженерія програмного забезпечення освітньо-професійної програми «Технології цифрового розвитку»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Анастасія СИТНІК
(підпис)

Виконав: здобувач вищої освіти групи ТЦР-44

Анастасія СИТНІК

Керівник: Ірина ГНІДЕНКО
старший викладач
Рецензент: _____

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут інформаційних технологій

Кафедра Технологій цифрового розвитку
Ступінь вищої освіти Бакалавр
Спеціальність 121 Інженерія програмного забезпечення
Освітньо-професійна програма «Технології цифрового розвитку»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Технологій цифрового розвитку
_____ Вікторія ЖЕБКА
«___» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Ситнік Анастасії Едуардівні

1. Тема кваліфікаційної роботи: «Web-сервіс для автоматичної генерації звітів та документів за шаблоном» керівник кваліфікаційної роботи старший викладач кафедри Ірина ГНІДЕНКО, затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 р. № 51.
2. Строк подання кваліфікаційної роботи: «18» травня 2026 р.
3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про методи автоматизації документообігу, опис роботи веб-додатків для генерації файлів, документація мови Python та бібліотек python-docx, ReportLab.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1. Аналіз предметної області автоматизації документообігу та дослідження існуючих аналогів.
2. Огляд та обґрунтування засобів реалізації веб-сервісу для маніпуляції даними.

3. Проектування архітектури системи та механізмів динамічного формування таблиць.
4. Програмна реалізація та тестування функціоналу генерації документів DOCX/PDF.
5. Перелік графічного матеріалу: *презентація*
    1. Порівняльний аналіз аналогів.
    2. Функціональні вимоги до системи.
    3. Стек технологій та бібліотек.
    4. Діаграма варіантів використання (Use Case).
    5. Схема архітектури веб-сервісу.
    6. Діаграма класів та методів генерації.
    7. Екранні форми веб-інтерфейсу.
    8. Зразки згенерованих документів.

6. Дата видачі завдання: «20» лютого 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	25.02.-04.03.2026	Виконано
2	Аналіз предметної області та існуючих аналогів	05.03.-13.03.2026	Виконано
3	Обґрунтування ІТ-засобів та технологій розробки	14.03.-16.03.2026	Виконано
4	Проектування архітектури та механізмів маніпуляції даними	17.03.-01.04.2026	Виконано
5	Програмна реалізація веб-сервісу та генератора файлів	02.04.-15.04.2026	Виконано
6	Тестування функціоналу та аналіз результатів	16.04.-18.04.2026	Виконано
7	Оформлення роботи: вступ, розділи, висновки	25.04.-29.04.2026	Виконано
8	Розробка демонстраційних матеріалів (презентація)	10.05.-11.05.2026	Виконано
9	Попередній захист роботи	12.05.2026	Виконано

Здобувач вищої освіти _____Анастасія СИТНІК

Керівник кваліфікаційної роботи _____ Ірина ГНІДЕНКО

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи містить: 70 стор., 15 рис., 5 табл., 18 джерел.

Мета роботи – підвищення операційної ефективності малих підприємств та підрозділів шляхом розробки інструменту для автоматизованого створення документації на основі введених даних.

Об'єкт дослідження – процедури створення, перевірки та фінального вивантаження цифрових документів у сучасних корпоративних системах.

Предмет дослідження – програмні інструменти та математично-алгоритмічний базис для гнучкого формування файлів у форматах PDF і DOCX за допомогою мови програмування Python.

Короткий зміст роботи: Для досягнення поставленої мети розроблено вебплатформу, яка автоматизує процеси формування вихідних документів. У рамках кваліфікаційної роботи доведено практичну користь від переведення документообігу підприємств малого бізнесу в автоматичний режим. Серед головних результатів інженерної розробки виділено: оригінальний алгоритм циклічного копіювання рядків, який будує динамічні таблиці із збереженням вихідної розмітки; усунення збоїв під час відображення кириличних символів у PDF-файлах завдяки примусовій інтеграції шрифтових гліфів UTF-8; багаторівневу систему фільтрації та валідації користувацьких даних для мінімізації критичних помилок під час генерації. Розроблене програмне рішення дозволяє отримувати готові стандартизовані документи в реальному часі.

КЛЮЧОВІ СЛОВА: ЦИФРОВІЗАЦІЯ ДОКУМЕНТООБІГУ, АВТОМАТИЧНЕ СТВОРЕННЯ ДОКУМЕНТІВ, PYTHON, ВЕБПЛАТФОРМА, ГНУЧКИЙ ШАБЛОН, DOCX, PDF, ОБРОБКА ДАНИХ, ХМАРА RENDER, ШРИФТОВІ ГЛІФИ.

ЗМІСТ

ВСТУП

Актуальність: В умовах стрімкої цифрової трансформації бізнес-процесів, ефективне управління документообігом стає фундаментальною складовою успішної діяльності будь-якого підприємства. Для малих консалтингових фірм та HR-відділів питання оперативної підготовки договорів, актів та інших юридично значущих документів є критичним. З огляду на постійне зростання обсягів інформації, що потребує документування, питання мінімізації рутинної ручної праці та виключення помилок, спричинених людським фактором, стає надзвичайно актуальним.

Сучасні веб-технології відкривають широкі можливості для оптимізації адміністративних процесів, дозволяючи створювати інструменти для миттєвої маніпуляції даними та автоматичної генерації вихідних файлів за заданими шаблонами. Одним із найбільш ефективних підходів є розробка спеціалізованих веб-сервісів, які інтегрують зручний інтерфейс введення даних із потужними алгоритмами обробки на бекенді. Використання таких сервісів дозволяє стандартизувати вихідну документацію, забезпечити її відповідність нормативним вимогам (наприклад, GDPR) та значно скоротити час на підготовку типових документів. Це створює для бізнесу простір для фокусування на стратегічних завданнях, делегуючи технічне формування файлів автоматизованим алгоритмам.

Розробка веб-сервісу для автоматичної генерації звітів та документів мовою програмування Python надає користувачам можливість швидко трансформувати мінімальний набір вхідних даних у професійно оформлені файли форматів PDF та DOCX, забезпечуючи високу гнучкість у роботі з динамічними структурами, такими як складні таблиці послуг.

Об’єктом дослідження є процеси автоматизації формування та маніпуляції електронними документами у корпоративному середовищі.

Предметом дослідження є засоби програмної реалізації веб-сервісу мовою Python для автоматичної генерації стандартизованих документів за шаблонами.

Метою роботи є підвищення операційної ефективності малих підприємств та підрозділів шляхом розробки інструменту для автоматизованого створення документації на основі введених даних.

Методи дослідження: першочергово було проаналізовано існуючі програмні рішення та сервіси автоматизації документообігу для формування функціональних та нефункціональних вимог до власного програмного забезпечення. Наступним кроком є проведення огляду бібліотек Python та хмарних платформ (зокрема Render), які відповідають вимогам щодо швидкодії, підтримки кириличного кодування та безпеки обробки даних.

Наукова новизна роботи визначається наступними функціональними складовими: реалізація алгоритму динамічної ін’єкції даних у складні табличні структури шаблонів та забезпечення крос-форматної підтримки (PDF/DOCX) з повною сумісністю UTF-8 для коректного відображення локалізованого контенту.

Практична значущість результатів полягає у можливості впровадження розробленого веб-сервісу в роботу консалтингових фірм та відділів кадрів, що дозволяє автоматизувати створення актів, договорів та звітів, знижуючи часові витрати на адміністрування на 60–80%.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області та проблематика автоматизації документообігу

Функціонування сучасних комерційних структур, установ та корпоративних об'єднань нерозривно пов'язане із супроводом, обробкою та систематичним відтворенням значних масивів текстової, юридичної та звітної інформації. У щоденній практичній діяльності суб'єктів малого підтриємства, кадрових підрозділів, правових департаментів та невеликих консалтингових об'єднань виникає нагальна потреба в оперативній підготовці значного спектра уніфікованих, стандартизованих та юридично значущих документів. До цієї категорії належать цивільно-правові договори, акти прийому-предачі наданих послуг або виконання робіт, рахунки-фактури, офіційні відповіді на зовнішні та внутрішні запити, розпорядча документація, кадрові накази, довідки та заявки.

Специфіка розглянутої предметної області характеризується присутністю комплексу деструктивних факторів, які суттєво гальмують операційну діяльність організацій, знижують загальну продуктивність праці адміністративного персоналу та зумовлють додаткове фінансове навантаження. Серед ключових деструктивних явищ слід приділити увагу високій ймовірності виникнення технічних та помилок у значенні слів, зумовлених безпосереднім впливом людського фактору. Під час виконання рутинних операцій з пернесенням реквізитів, текстових блоків, ідентифікаційних даних та цифрових показників із внутрішніх облікових реєстрів чи систем класу CRM до статистичних локальних файлів-шаблів, фахівці часто припускаються механічних помилок, описок та викривлень інформації. Спотворення імен клієнтів, назв контрагентів, платіжних реквізитів, дат або вартісних параметрів, створює суттєві правові загрози для бізнесу, призводить до визнання угод недійсними

та вимагає повторного проходження й перевипуску цілого пакету супровідних документів.

Додатковим негативним аспектом виступають непродуктивні часові витрати кваліфікованих співробітників на виконання одноманітних, повторюваних операцій. Процеси пошуку актуальних версій регуляторних бланків на локальних носіях, їх ручне заповнення в тестових редакторах, перевірка дотримання правил корпоративного стилю розширення, ручне форматування табличних областей та фінальна конверсія у захищені формати для відправки контрагентам займають значну частину робочого дня. У масштабах місячного або річного операційного циклу підтриємства зазначені втрати акумулюються у десятки людино-годин, які доцільніше було б спрямувати на вирішення складних аналітичних, стратегічних чи управлінських завдань.

Ситуація значно ускладнюється у процесах оперування динамічними структурами даних. Найбільш критичною та трудомісткою процедурою є конкретне заповнення табличних зон, наприклад, під час формування актів або звітів, що містять розгалужений перелік наданих послуг із різними вартісними, кількісними та податковими характристиками. Наявні статичні заготовки документів у поширених офісних пакетах не володіють адаптивними інструментами для автоматичного масштабування, де будь-яке примусове ручне коригування структури таблиці призводить до небажаного зсуву елементів дизайну, порушення пропорцій, появи незаповнених зон та викривлення фінального візуального вигляду всього документа.

Окрема частина технічних складнощів формують питання міжплатформної сумісності та мовної локалізації. Забезпечення ідентичного рендерингу згенерованого документта на обчислювальних машинах під керуванням різних операційних систем зазвичай є новаторським завданням через відмінності у системних шрифтах та алгоритмах відображення текстових блоків. Окрім того, процеси конвертації контенту в PDF-формат часто супроводжується критичними збоями під час обробки

кириличних символів. За відсутності інтегрованого гнучкого налаштування сумісності з UTF-8 це призводить до псування кінцевого файла, заміни букв хаотичними нечитабельними знаками або повної втрати структури тексту.

Усі перелічені чинники та інженерні проблеми зумовлюють об’єктивну необхідність створення легковагової, кросплатформеної та інтуїтивно зрозумілої вебплатформи. Таке програмне рішення має дозволяти оператору вносити виключно ключові змінні параметр через інтерактивні форми інтерфейсу, делегуючи завдання обробки даних, динамічної побудови внутрішньої структури та миттєвої генерації готового файла автоматизованим серверним алгоритмам.

Впровадження подібного вебсервісу дозволить представникам малого бізнесу та стандартам успішно реалізувати концепцію повністю цифрового офісу без значних капітальних інвестицій на придбання, розгортання та довготривалу адаптацію важких і дорогих корпоративних ERP-систем.

1.2. Специфіка автоматизації процесів генерації та маніпуляції даними

Комплексна автоматизація процедур формування вихідних документів є складним багатоетапним процесом, який не може бути зведений до механічного копіювання чи підстановки текстових фрагментів. Цей процес охоплює взаємодію багатьох іольованих програмних модулів, кожен з яких вирішує специфічні завдання валідації, трансформації, рендерингу та безпечного збереження інформаційних потоків.

1.2.1. Гнучкість структури та маніпуляція динамічними даними

Головним викликом на етапах проєктуання та езпосередньої програмної реалізації логіки вебсервісу є організація коректної обробки вхідних інформаційних масивів, обсяг і структура яких не є відомими заздалегідь.

Реальні комерційні, кадрові чи звітні документи практично ніколи не мають фіксованого обсягу полів і регулярно містять таблиці з довільною кількістю позицій, найменувань або розрахункових елементів. Для подолання цього обеження виникає інженерна потреба в розробці адаптивного алгоритму модифікації даних, здатного здійснювати динамічне розширення та стиснення табличних сіток у внутрішній структурі шаблонів. Алгоритм повинен повінстю унеможливлювати появу візуальних дефектів початкової верстки, наприклад, неконтрольовані розриви сторінок, накладання ліній або вихід тексту за межі друкованих полів.

Розроблене програмне забезпечення має також підтримувати автоматизацію супутніх математичних та логіко-мовних обчислень. Система пови нна самостійно хдійснювати калькуляцію підсумкових сум, розраховувати відсоткові податкові нарахування та виконувати операцію проведення цифрових значень у текстовий формат (сума прописом) відповідно до граматичних і синтасичних правил цільової мови локаізації.

Важливим бар’єром на шляху забезпечення стабільності вебзастосунку є впровадження суворих багаторівневих правил попередньої перевірки (валідації) та очищення (санітизації) всіх параметрів, що надходять від клієнтського інтерфейсу. Реалізація компонентів валідації типу типів, довжини та структури вхідних даних дозволяє ефективно блокувати некоректно сформовані запити на ранніх етапах обробки, гарантуючи абсолютний захист внутрішньої XML архітектури документів від пошкоджень та забезпечючи безвідмовне функціонування сервісу під час ін’єкції даних у складні структури шаблонів.

1.2.2. Проблематика багатоформатності (DOCX vs PDF)

Для забезпечення повноцінного та всеохопного корпоративного документообігу розроблювана система має підтримувати паралельну роботу з двома принципово

відмінними форматами електронних даних, які вирішують комплементарні завдання в бізнес-процесах організації.

Формат DOCX є домінуючим стандартом на етапах первинного проєктування, редагування та узгодження текстового контенту. Його використання надає користувачам гнучкі можливості для внесення локальних правок, додавання пунктів, зміни формулювань або проведення спільного двостороннього рецензування угод із залученням контрагентів та юридичних служб. З позицій програмної інженерії, створення модуля генерації файлів DOCX вимагає побудови інструментів для прямого керування внутрішньою архітектурою файлого контейнера OpenXML. Програмні алгоритми мають виконувати точкову заміну заздалегідь визначених семантичниз тегів-заповнювачів на текстові масиви, що надійшли з бази даних чи вебплатформи, без руйнування стилістичних карт документа.

Водночас критично важливою функцією є забезпечення експорту сформованих документів у нередагований формат PDF. Цей стандарт є обов’язковою вимогою для створення фінальних копій документів, призначених для накладання цифрових підписів, офіційного надсилання та довготривалого архівного зберігаеея у корпоративних середовищах. Проте процес рендерингу PDF-документів засобами базових бібліотек програмування супроводжується проблемою сумісності з регіональними кодуваннями. Більшість стандартних інструментів генерації графічних та текстових потоків за замовчуванням орієнтовані на латинський алфавіт, через що спроба виведення кириличних символів призводить до появи критичних помилок відображення (ефект «тофу»), коли замість літер відображаються порожні прямокутники. Для подолання цієї вразливості архітектура вебсервісу має передбачати обов’язкове динамічне вбудовування зовнішніх TrueType-шифрованих глыіфів безпосередньо у структуру вихідного файлу. Такий підхд забезпечує повну кросплатформеність та гарантує стовідсотково ідентичний візуальний вигляд, точне

збереження геометрії та коректне кодування контенту, незалежно від програмного чи апаратного середовища кінцевого користувача.

1.2.3. Масштабованість та хмарне середовище

Проєктування вебплатформи за моделлю Saas зумовлює необхідність забезпечення високої відмовостійкості, швидкої реакції на запити та низького рівня споживання ресурсів в ізольованих умовах сучсних хмарних інфраструктур (зокрема на платформі Render). Це вимагає проведення глибокої оптимізації алгоритмів бекенд-частини застосунку, мінімізації накладних обчислювальних витрат та суворого контролю за виділенням оперативної пам’яті (RAM). Запобігання витокам пам’яті та оптимізація скриптів є життєво необхідними для збереження стабільності платформи під час пікових навантажень, коли велика кількість користувачів одночасно ініціює створення об’ємних або графічно складних документів.

Іншим важливим аспектом є правильна конфігурація механізмів оперування тимчасовими файловими об’єктами на стороні сервера. Логіка функціонування системи має передбачити, що всі операції з наповнення та трансформації файлів виконуються динамічно в момент обробки поточного HTTP-запиту. Сформований документ повинен миттєво транслювати клієнту у вигляді байтового потоку через веб-інтерфейс і відразу після успішного завершення сесії завантаження автоматично та безповоротно видалятися з дискового простору хмарного сервера. Такий підхід не лише оптимізує використання обмеженого накопичувача, але й виступає юазовим критерієм побудови безпечних вебсистем, запобігаючи несанкціонованому накопиченню конфіденційного контенту на серверній стороні.

1.2.4. Питання безпеки та відповідності стандартам (GDPR)

Оскільки процеси автоматичного заповнення кадрових контрактів, фінансових звітів або клієнтських угод пов'язані з обробкою конфіденційних відомостей, питання

інформаційної безпеки та відповідності міжнародним регуляторним стандартам (злкрема, Закону України «Про захист персональних даних» та Загальноєвропейському регламенту про захист даних - GDPR) посідають чільне місце в архітектурі системи.

Для забезпечення повного комплаєнтсу з принципом обмеження зберігання даних, архітектурні рішення вебсервісу мають гарантувати, що всі маніпуляції, семантичний аналіз та ін’єкції інформації здійснюються виключно всередині легкої оперативної пам’яті сервера. Категорично забороняється проміжне кешування, створення тимчасових незашифрованих копій на локальних дискових масивах або логування персональних атрибутів у текстові журнали сервера. Програмні алгоритми обробки GDPR-запитів повинні забезпечувати абсолютну точність перенесенян даних у вихідні бланки, викликаючи будь-яку ймовірність витоку, підміни чи випадкової модифікації чутливих даних у процесі формування кінцевого документа.

1.2.5. Користувацький досвід (UX) у технічному середовищі

Орієнтація розробленого програмного продукту на представиків мікробізнесу, менеджерів та адміністраторів вимагає врахування фактора присутності у кінцевих операторів соціальних технічних навичок або досвіду у сфері програмування. Через це складна математична, алгоритмічна та інфраструктурна логіка маніпуляції XML-структурами файлів має бути повністю абстрагована та прихована від користувача за лаконічним, адаптивним та інтуїтивно зрозумілим графічним інтерфейсом.

Ключовим елементом підвищення якості користувацького досвіду (UX) є спрощення взаємодії з динамічними структурами. Клієнту не потрібно самостійно вираховувати координати елементів чи вручну додавати розмітку всередині текстового редактора. Йому достатньо ініціювати команду створення нового рядка в таблиці натискання однієї інтерактивної кнопки веб-інтерфейсу, після чого внутрішні алгоритми бекенду самостійно перерахують геометричні параметри шаблону,

адаптують межі сумужних блоків та коректно інтегрують нові змінні поля, забезпечуючи високу швидкість та комфорт виконання повсякденних операцій.

1.3. Дослідження існуючих аналогів та порівняльний аналіз

Сучасний ринок програмного забезпечення у сфері LegalTech та HR-технологій представлений багатьма інструментами для автоматизації процесів документообігу. Проте переважна більшість наявних платформ розробляється з орієнтацією на великий корпоративний сегмент, що ускладнює їхнє впровадження в межах малого бізнесу. Для визначення оптимальних технологічних рішень доцільно проаналізувати найбільш релевантні системи, що функціонують на ринку.

Серед міжнародних платформ одним із найпопулярніших рішень для створення, трекінгу та підписання електронних документів є сервіс PandaDoc. Ця система оснащена потужним вбудованим візуальним редактором і підтримує глибоку інтеграцію даних із різноманітними CRM-платформами. Попри значний функціонал, використання PandaDoc у діяльності мікробізнесу обмежене через високу вартість ліцензійної підписки. Крім того, інтерфейс системи є занадто перевантаженим для виконання простих або разових операцій, а адаптація специфічних локальних шаблонів документів вимагає тривалого попереднього налаштування.

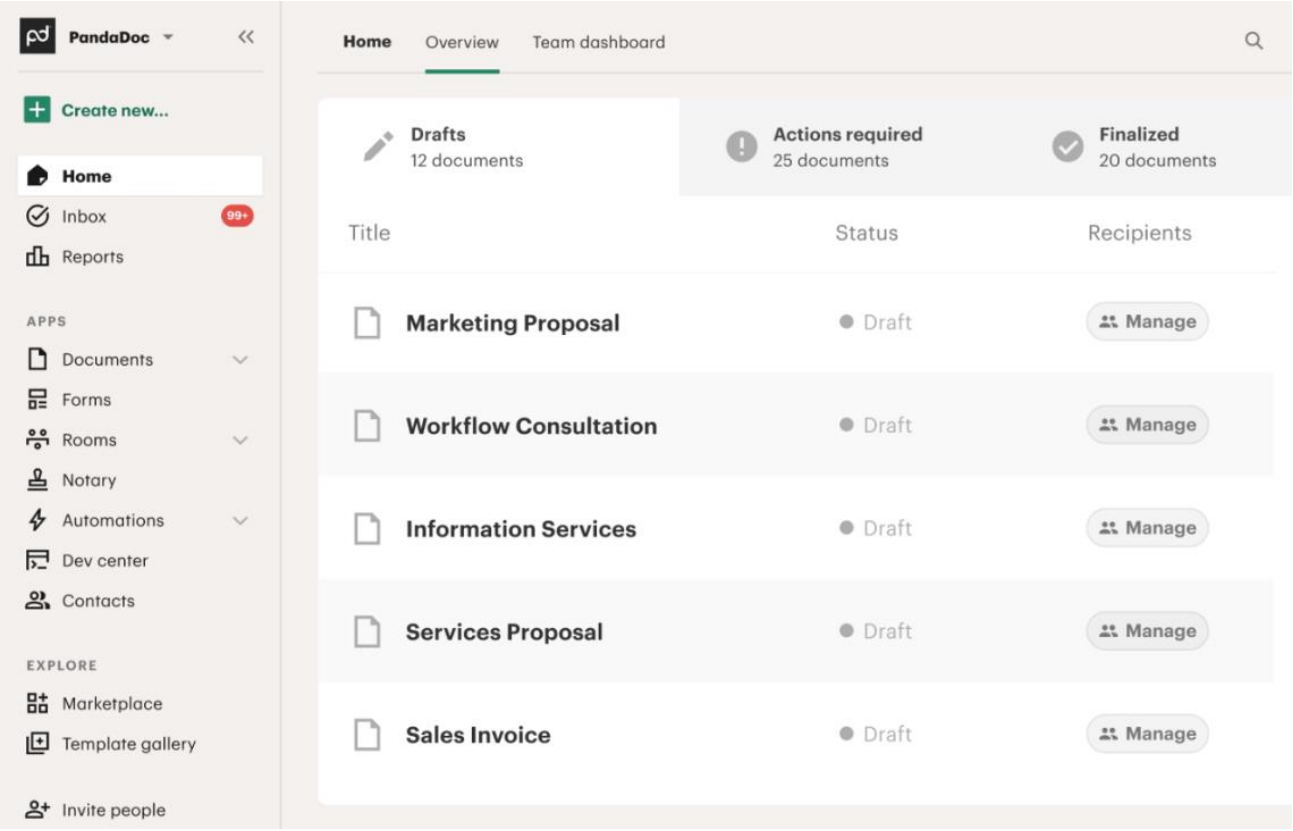


Рис. 1.1 Інтерфейс користувача сервісу PandaDoc для управління шаблонами та документами

Продовжуючи огляд існуючих аналогів, варто розглянути систему DocuSign, зокрема її модуль Gen для Salesforce. Ця платформа є світовим лідером у сегменті електронного підписання та пропонує інструменти для автоматичного заповнення текстових полів у контрактах безпосередньо перед їхнім завіренням.

Головним обмеженням цього рішення є його жорстка прив'язка до екосистеми Salesforce, що робить сервіс недоступним або економічно недоцільним для компаній, які не використовують цю CRM-платформу. Крім того, функціональний фокус системи зміщений у бік верифікації та захисту підписів, а не гнучкої маніпуляції складними динамічними таблицями, тоді як процес початкового технічного налаштування вимагає високого рівня кваліфікації адміністраторів.

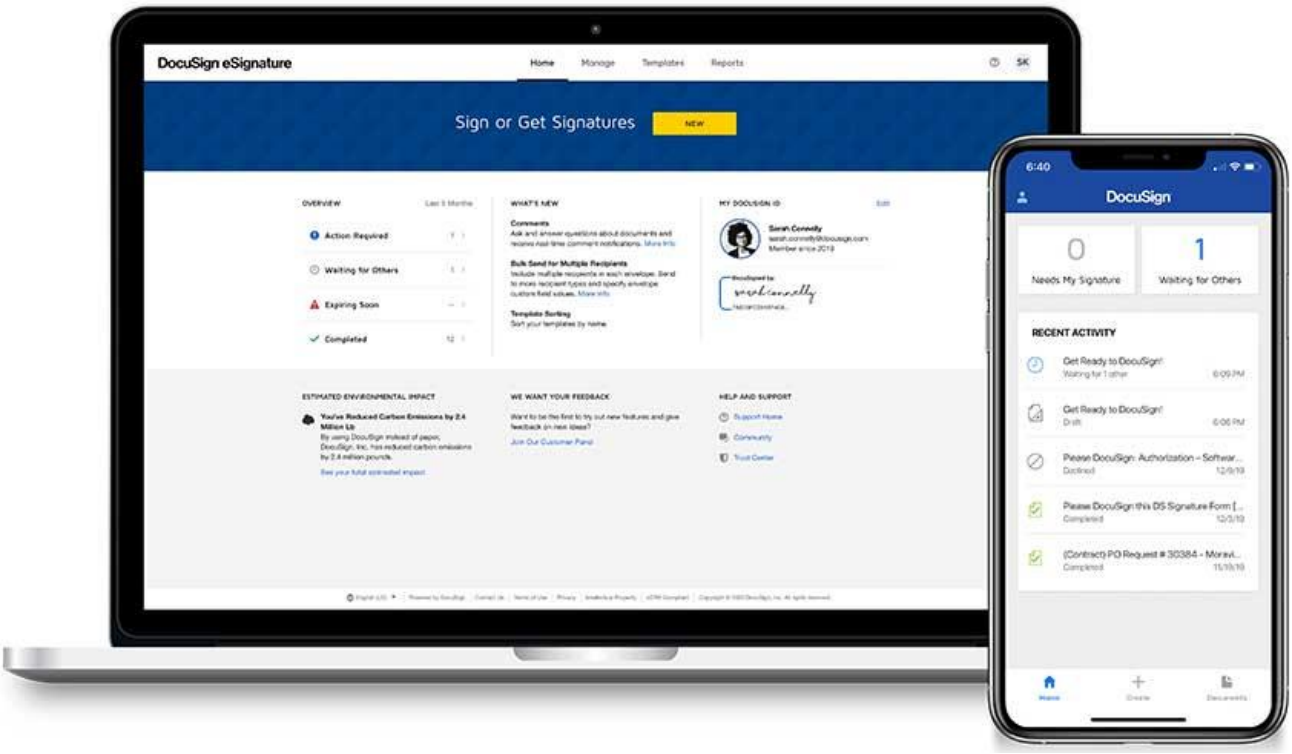


Рис. 1.2 Інтерфейс сервісу DocuSign на десктопних та мобільних пристроях

Іншим спеціалізованим інструментом є платформа HotDocs, яка позиціонується як професійне рішення для автоматизації корпоративної та юридичної документації. Цей сервіс активно застосовується великими правовими департаментами для побудови складних логічних послідовностей та розгалужених сценаріїв у текстах угод.

Проте практичне використання HotDocs у сегменті малого бізнесу ускладнюється через морально застарілий інтерфейс користувача та надлишкову архітектурну складність, яка є невиправданою для оперативного формування базових звітів чи актів виконаних робіт. Додатковим деструктивним чинником виступає високий поріг входження, що змушує компанії витрачати додаткові ресурси на спеціальне тривале навчання персоналу перед початком експлуатації платформи.

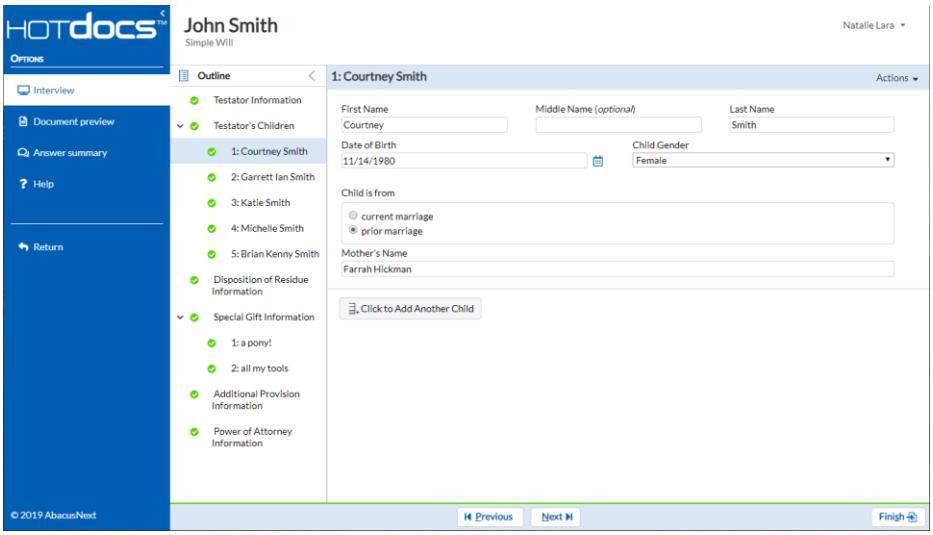


Рис. 1.3 Інтерфейс збору даних у системі HotDocs для автоматизації складних шаблонів

Останнім серед найважливіших конкурентних рішень варто виділити AxDraft – вітчизняний продукт, створений спеціально для автоматизації підготовки юридичної та супровідної документації. Платформа надає зручні інструменти для конструювання угод з окремих стандартизованих текстових блоків, що значно прискорює роботу з типовими контрактами.

Попри очевидні переваги локального продукту, AxDraft розробляється з чітким фокусом на великі юридичні департаменти та enterprise-сегмент. Для невеликих HR-відділів чи агенцій малого бізнесу використання системи ускладнюється через відсутність доступних бюджетних тарифних планів (low-cost) або безкоштовних тестових версій. Крім того, закрита екосистема платформи обмежує можливості швидкої сторонньої інтеграції без залучення технічної підтримки провайдера.

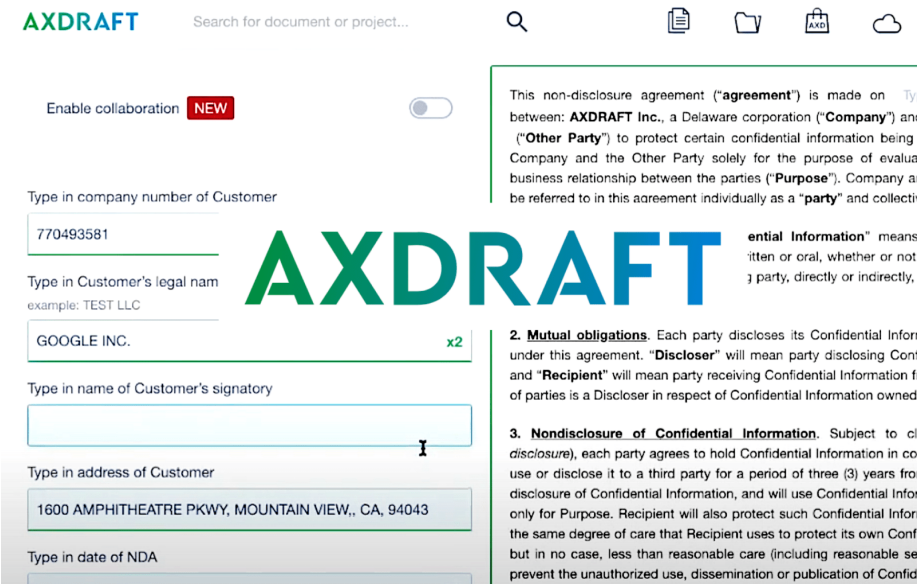


Рис. 1.4 Інтерфейс автоматизованого заповнення юридичних документів у сервісі AXDRAFT

Таблиця 1.1

Порівняльна характеристика засобів автоматизації

Критерій порівняння	PandaDoc	DocuSign	HotDocs	Розроблений сервіс
Складність інтерфейсу	Висока (багатофункціональний)	Середня	Висока (професійний)	Низька (мінімалістичний)
Маніпуляція таблицями	Обмежений базовий інструментарій	Шаблонна генерація	Складна внутрішня логіка	Динамічна обробка на базі Python
Підтримка кирилиці	Повна	Повна	Часткова	Повна (кодування UTF-8)
Фінансова модель	Висока вартість підписки	Висока вартість підписки	Корпоративні ліцензії	Відкритий код (Open Source)
Швидкість впровадження	Потребує тривалого налаштування	Середній термін адаптації	Потребує попереднього навчання	Негайне розгортання

Актуальність розробки. Проведений аналіз наявного програмного забезпечення свідчить, що більшість сучасних ринкових продуктів характеризується або занадто високою вартістю ліцензій, або надлишковим інструментарієм, який суттєво уповільнює операційну діяльність представників малого консалтингу. Пропонований у цьому дослідженні вебсервіс спрямований на заповнення ніші так званих легковагових систем автоматизації (Lightweight Document Automation). Ефективність такого рішення досягається завдяки максимальному спрощенню процедури введення інформації оператором, реалізації функцій динамічного керування кількістю рядків у табличних структурах, що зазвичай є слабким місцем статичних шаблонізаторів, а також можливості миттєвого завантаження готового файла без потреби в інтеграції та реєстрації користувача у громіздких CRM-платформах. Наведені технологічні переваги забезпечують високу практичну значущість кваліфікаційної роботи, орієнтованої на задоволення реальних потреб локального ринку малих підприємств та індивідуальних підприємців в Україні.

1.4. Обґрунтування вибору ІТ-засобів та технологій реалізації

Створення ефективного вебсервісу для автоматизованого формування документів вимагає застосування технологічного стеку, здатного забезпечити високу швидкість обробки інформації та гнучкість налаштувань. Основним інструментом розробки було обрано мову програмування Python, яка має потужний інструментарій для структурної маніпуляції даними.

1.4.1. Мова програмування Python

Вибір Python як базової мови програмування зумовлений її широкими можливостями у сфері автоматизації рутинних процесів та аналізу текстових масивів. Головна перевага цієї технології в межах кваліфікаційної роботи полягає у наявності

розвиненої екосистеми спеціалізованих модулів і бібліотек. Наявні рішення дозволяють організувати ефективну низькорівневу взаємодію з внутрішньою структурою файлів форматів DOCX та PDF.

Важливим фактором на користь обраної мови є висока швидкість написання та відлагодження вихідного коду. Лаконічний синтаксис та динамічна типізація дозволяють оперативно реалізувати складну логіку для циклічної обробки табличних елементів шаблонів без надмірного ускладнення архітектури застосунку. Крім того, повна кросплатформеність мови гарантує стабільне й швидке розгортання вебсервісу в ізольованих контейнерах на Linux-серверах хмарної платформи Render, забезпечуючи високу мобільність та потенціал для подальшого масштабування програмного продукту.

1.4.2. Вебфреймворк (FastAPI / Flask)

Побудова серверної частини розроблюваного застосунку базується на використанні легковагових мікрофреймворків, здатних підтримувати високу швидкість обробки клієнтських запитів. Основним інструментом було обрано FastAPI. Цей сучасний фреймворк забезпечує високу продуктивність завдяки вбудованим механізмам асинхронного програмування на основі стандарту ASGI. Для розробника суттєвим плюсом є автоматичне створення інтерактивної документації за стандартом OpenAPI (Swagger). Наявність такого інструменту дозволяє швидко протестувати працездатність кінцевих точок програми (endpoints) без написання додаткових скриптів для верифікації запитів.

Логіку динамічного заповнення полів реалізовано за допомогою двигуна шаблонізації Jinja2. Попри його традиційне застосування у вебсервісах для рендерингу клієнтського HTML-інтерфейсу, у цій роботі інструментарій адаптовано під інші завдання. Jinja2 виконує роль ядра для семантичної ін’єкції та зв’язування

користувацьких змінних із внутрішніми структурами файлів-шаблонів, гарантуючи точність передачі текстових даних у підсумковий документ.

1.4.3. Спеціалізовані бібліотеки для генерації файлів

Безпосередній процес формування вихідних документів та взаємодії з їхньою внутрішньою структурою реалізовано за допомогою набору спеціалізованих інструментів мови Python. Для опрацювання текстових файлів формату Microsoft Word інтегровано бібліотеку python-docx. Цей інструмент надає низькорівневий програмний доступ до окремих параграфів, табличних сіток та ізольованих комірок наявного шаблону. Використання її функціоналу дозволило створити алгоритм динамічного копіювання та наповнення рядків під час генерації актів приймання-передачі чи комерційних рахунків.

Експорт захищених копій у формат PDF забезпечується засобами бібліотек ReportLab або WeasyPrint. Платформа ReportLab дозволяє здійснювати точне позиціювання графічних та текстових елементів на сторінці. Для подолання проблеми некоректного рендерингу кириличних символів у межах розробленої архітектури застосовано метод примусової реєстрації зовнішніх TrueType-шрифтів за допомогою вбудованого модуля TTFont. Таке рішення повністю усуває ризик спотворення українського тексту в готових файлах.

Додатково до технологічного стеку залучено бібліотеку Pandas. Попри її традиційне призначення для аналізу великих масивів даних (Data Science), у межах цього проєкту інструмент адаптовано для попереднього очищення, структурування та валідації вхідної інформації. Використання Pandas є доцільним у випадках, коли користувач імпортує розгалужені списки клієнтів або переліки наданих послуг безпосередньо з таблиць формату Excel, що дозволяє швидко перетворити зовнішні файли на структуровані масиви для подальшої ін’єкції в шаблони.

1.4.4. Хмарна платформа Render

Хостинг та фінальне розміщення розробленого вебсервісу реалізовано на базі хмарної платформи Render, яка добре підходить для розгортання невеликих SaaS-застосунків. Одним із ключових чинників вибору цього середовища стала наявність інструментів автоматичного деплою (Auto-deploy). Цей механізм синхронізує сервер із репозиторієм на GitHub, ініціюючи оновлення робочої версії програми одразу після фіксації нових змін у коді. Такий підхід значно спрощує процес відлагодження системи у реальному часі.

Платформа має повну нативну підтримку мови Python. Завдяки цьому інсталяція задіяних у проєкті зовнішніх бібліотек та модулів відбувається автоматично під час збирання проєкту на основі стандартного конфігураційного файла requirements.txt, мінімізуючи ризики виникнення конфліктів залежностей.

Оскільки вебсервіс оперує конфіденційними даними клієнтів, особливу увагу приділено питанням інформаційної безпеки. Платформа Render за замовчуванням забезпечує генерацію та автоматичне продовження захищених SSL-сертифікатів для вебдодатків. Це дозволяє організувати передачу інформації виключно через шифрований протокол HTTPS, гарантуючи захист персональних відомостей від перехоплення чи стороннього втручання в процесі генерації документів.

1.4.5. Фронтенд-технології

Розробка клієнтської частини (фронтенду) вебсервісу орієнтована на створення мінімалістичного та швидкого інтерфейсу, що забезпечує оперативну взаємодію користувача з платформою. Візуальну структуру та стилізацію елементів реалізовано за допомогою сучасних стандартів HTML5 і CSS3. Це дозволило спроектувати адаптивні й кросбраузерні форми для введення первинної інформації, які коректно масштабуються під мобільні пристрої, планшети та монітори з великою роздільною здатністю екрана.

Динамічну поведінку сторінки та обмін даними з сервером забезпечує мова програмування JavaScript. Замість важких клієнтських фреймворків у проєкті використано чистий код (Vanilla JS) у зв'язці з бібліотекою Axios, що оптимізує швидкість завантаження вебсторінки. Цей інструментарій відповідає за відправку асинхронних запитів (AJAX) на бекенд-частину застосунку в момент, коли користувач підтверджує генерацію бланка. Перехоплюючи відповідь сервера, скрипт миттєво надає пряме посилання на скачування сформованого документа. Весь процес маніпуляції даними та отримання результату відбувається без перезавантаження сторінки, що значно покращує загальний користувацький досвід.

1.5. Об’єкт, предмет, мета та постановка завдань кваліфікаційної роботи

Результати аналізу предметної області, дослідження ринкових аналогів та оцінки сучасних технологічних засобів дозволили сформувати концептуальний базис цього дослідження.

Об’єктом дослідження виступають процеси формування, верифікації та експорту електронної документації в умовах цифровізації корпоративного середовища.

Предметом дослідження є алгоритмічне забезпечення та програмні засоби динамічної генерації файлів форматів PDF та DOCX із використанням мови програмування Python.

Мета роботи полягає в підвищенні ефективності процесів формування юридично значущої документації в консалтинговій діяльності шляхом розробки вебсервісу для автоматизованої генерації документів на основі динамічних вхідних даних та алгоритмів ітераційної обробки DOCX-шаблонів.

Для реалізації визначеної мети в межах кваліфікаційної роботи було сформульовано низку послідовних дослідницьких та інженерних завдань. Спочатку

необхідно дослідити специфіку корпоративного документообігу в сегменті малого бізнесу та визначити операційні процеси, які потребують першочергової модернізації. Наступний крок передбачає виконання критичного аналізу наявних комерційних систем автоматизації для виявлення їхніх функціональних та цінових обмежень під час інтеграції в малі підприємства. На основі отриманих висновків потрібно спроектувати загальну архітектуру майбутнього вебсервісу та описати модель взаємодії між клієнтським інтерфейсом і серверною частиною застосунку.

Важливим етапом дослідження є математичне та алгоритмічне обґрунтування логіки маніпуляції даними, зокрема створення алгоритму ітераційного клонування рядків для динамічного формування таблиць без порушення початкової верстки бланка. Практична частина роботи вимагає безпосередньої програмної реалізації бекенду мовою Python з інтеграцією модулів валідації користувацького вводу та впровадженням засобів коректного вбудовування гліфів UTF-8 для вирішення проблеми рендерингу кирилиці під час конвертації в PDF. Завершується цикл розробки розгортанням створеного вебзастосунку на хмарній платформі Render, проведенням серії тестів для оцінки стабільності системи за умов паралельних запитів та верифікацією відповідності розробленого графічного інтерфейсу вимогам зручності та простоти використання.

2 ПРОЕКТУВАННЯ ВЕБСЕРВІСУ ДЛЯ АВТОМАТИЗОВАНОЇ ГЕНЕРАЦІЇ ДОКУМЕНТІВ

Розроблювана система функціонує за класичною клієнт-серверною моделлю, де обробка логіки маніпуляції даними та взаємодія із шаблонами винесена на сторону сервера під керуванням мікрофреймворку Python. Інтерактивний фронтенд слугує для збору та первинної валідації введених користувачем атрибутів. Серверна частина містить обчислювальний модуль, який забезпечує покроковий рендеринг вихідних файлів форматів PDF та DOCX, динамічну перебудову табличних структур та інтеграцію шрифтових контейнерів.

2.1 Проектування загальної архітектури системи

Для забезпечення високої відмовостійкості, ізоляції окремих функціональних блоків та спрощення подальшого масштабування вебсервісу було застосовано модульний архітектурний підхід. Ключовим інженерним рішенням під час проектування стало чітке розмежування логіки представлення інтерфейсу користувача та логіки внутрішньої обробки інформаційних масивів.

Архітектура розробленого програмного продукту об'єднує чотири взаємопов'язані технологічні компоненти. Першим рівнем виступає модуль веб-інтерфейсу, що складається з набору адаптивних форм для збору вхідних параметрів. Наступним елементом є API-контролер на базі Python, який приймає вхідні JSON-об'єкти, здійснює їхню структурну перевірку на відповідність типам даних та перенаправляє потоки інформації до відповідних сервісів генерації.

Ядром системи є core-модуль маніпуляції шаблонами, побудований на базі інструментів python-docx та ReportLab. У межах цього компонента реалізовано

ключовий алгоритм ітераційного сканування рядків, який відповідає за динамічне розширення таблиць у документах. Фінальним елементом архітектури є тимчасове сховище хмарної платформи (Runtime Storage). Воно виділяється в оперативній пам'яті сервера Render і функціонує як ізольована буферна зона, де відбувається безпосереднє збирання та кодування файла перед його трансляцією у клієнтську сесію.

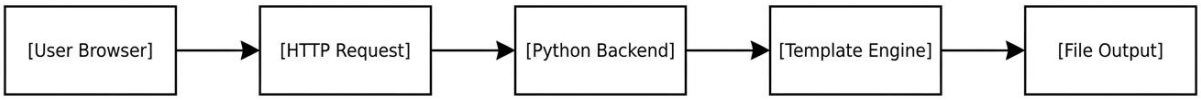


Рис. 2.1 Схема архітектури (Architecture Diagram).

2.2 Моделювання вимог до системи

Для декомпозиції та деталізації функціональних можливостей проектованого вебсервісу застосовано методологію об'єктно-орієнтованого аналізу з використанням стандартизованої нотацію UML, зокрема діаграм прецедентів (Use Case). Головним актором системи є кінцевий користувач – менеджер консалтингової агенції або спеціаліст кадрового відділу. Взаємодіючи з графічним інтерфейсом, цей актор ініціює сесію створення документа, визначає цільовий шаблон, вносить вихідні атрибути та завантажує сформований файл.

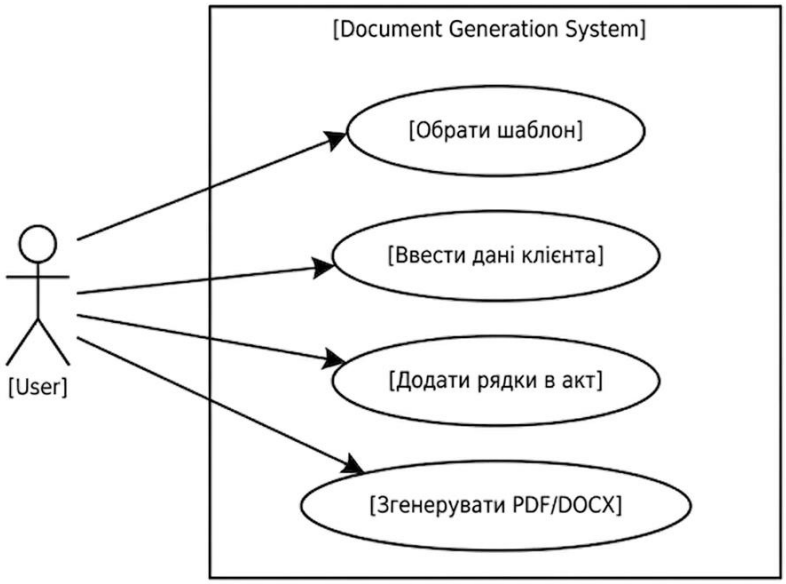


Рис. 2.2 Діаграма прецедентів (Use Case Diagram).

Аналіз вимог дозволив сформувати чіткий перелік функціональних завдань, які має вирішувати архітектура застосунку. Першочергово забезпечується інтерфейсна логіка для зручного введення текстових і числових констант через інтерактивні поля вебформи. Ключовою інженерною вимогою до системи є реалізація механізму динамічного розширення табличних масивів. Користувач повинен мати можливість без обмежень додавати нові позиції товарів чи послуг, не руйнуючи при цьому цілісність та пропорції фінального документа.

Програмна логіка гарантує автоматичну генерацію файлів на основі попередньо завантажених типових зразків договорів, актів приймання-передачі чи офіційних запитів. При цьому архітектура системи передбачає варіативність вихідних форматів. Залежно від операційних потреб користувач може експортувати документ у форматі DOCX для подальшого локального рецензування або у форматі PDF для архівного зберігання та друку. Нефункціональні вимоги до вебсервісу фокусуються на забезпеченні повної транслітерації та безпомилкового відображення кириличних символів українського алфавіту в обох типах вихідних контейнерів, що досягається за рахунок інтеграції шрифтових TrueType-пакетів на рівні сервера.

Таблиця 2.1

Зведена таблиця функціональних вимог для вебсервісу автоматичної генерації документів

№	Вимога	Опис
1	Вибір шаблону документа	Можливість обрання користувачем типу документа (договір, акт, звітність тощо) з наявного переліку в інтерфейсі.
2	Введення динамічних даних	Інтерактивна форма для заповнення текстових та числових параметрів (реквізити, дати, суми, персональні дані).
3	Динамічне керування таблицями	Підтримка додавання необмеженої кількості нових рядків у табличну структуру документа (зокрема, для списку послуг).
4	Формування вихідних файлів	Генерація кінцевих документів на основі внесених даних у форматах DOCX та PDF за заданими шаблонами.
5	Експорт та збереження	Завантаження сформованого документа безпосередньо на локальний пристрій користувача після завершення обробки.
6	Локалізація (підтримка кирилиці)	Забезпечення коректного кодування та відображення українського алфавіту при експорті документів у формат PDF.

Таблиця 2.2

Зведена таблиця нефункціональних вимог для вебсервісу автоматичної генерації документів

№	Вимога	Опис
1	Ергономічність (Usability)	Інтуїтивно зрозумілий інтерфейс користувача, що забезпечує введення даних без додаткової підготовки чи навчання.
2	Швидкодія (Performance)	Час генерації типового документа не перевищує 2–4 секунди після надсилання форми.
3	Безпека (Security)	Конфіденційність даних забезпечується архітектурою "on-the-fly": персональна інформація видаляється після завершення сесії.
4	Доступність (Availability)	Безперебійна робота сервісу в режимі 24/7 через веббраузер, без необхідності встановлення клієнтського ПЗ.
5	Масштабованість (Scalability)	Архітектура додатка підтримує розширення системи шляхом додавання нових шаблонів .docx у відповідну директорію сервера.

2.3 Проектування структури даних та шаблонів

З міркувань безпеки та забезпечення конфіденційності користувачів вебсервіс спроектовано за принципом обробки даних на льоту («on-the-fly»). Такий підхід виключає персистентне збереження персональної інформації у базах даних, тому ключовим етапом розробки став опис структури JSON-об'єкта, який передається між клієнтською та серверною частинами під час HTTP-запиту. Логічна архітектура цього об'єкта для генерації акта виконаних робіт містить кілька ієрархічних блоків, об'єднаних у єдиний пакет даних.

Початковий сегмент структури (Header) відповідає за метадані документа. Сюди входять унікальний номер акта, дата його реєстрації, а також константні відомості про компанію-виконавця, включаючи її назву та банківські реквізити. Наступний інформаційний блок (Customer) акумулює ідентифікаційні дані контрагента: найменування юридичної особи або ПІБ фізичної особи-підприємця, її адресу та податкові атрибути.

Найскладнішим з інженерного погляду є вкладений масив об'єктів (Items), який має динамічну довжину. Кожен елемент цього масиву описує окрему позицію у переліку наданих послуг чи відвантажених товарів. Структура елемента включає текстове поле з описом робіт, числове поле кількості, одиниці виміру та фіксовану вартість за одиницю. Завершує JSON-структуру підсумковий блок (Footer). Він містить агреговані показники, такі як загальна сума до сплати та розмір податкових відрахувань, які розраховуються автоматично на основі даних із масиву Items, а також службові поля для розміщення підписів і печаток сторін.

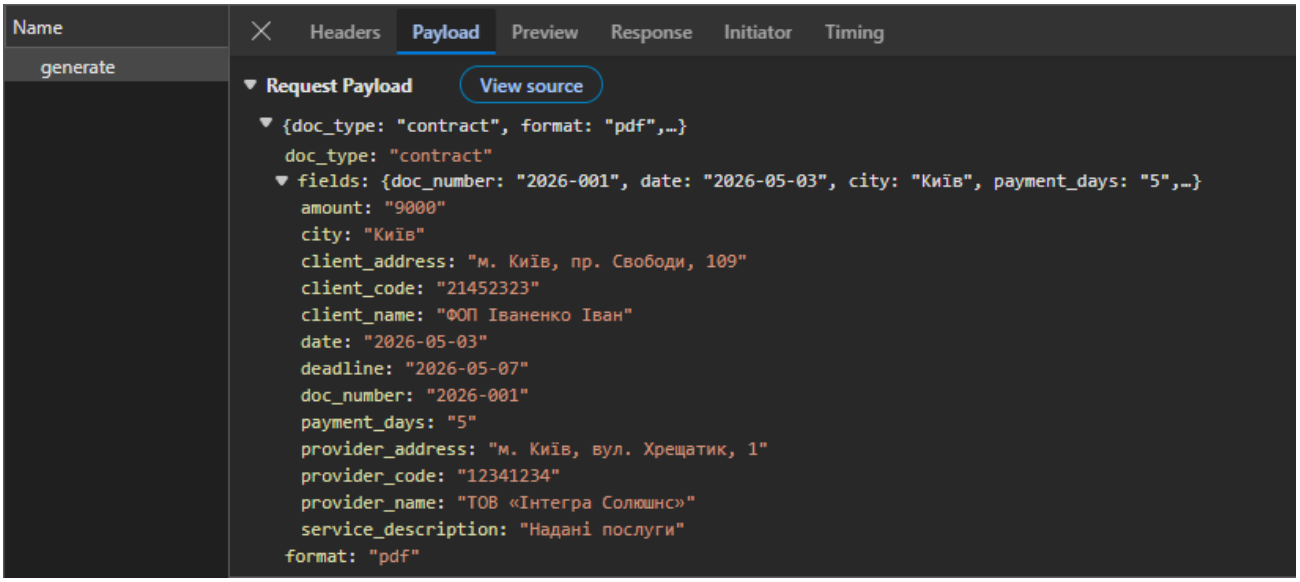


Рис. 2.3 Структура JSON-об’єкта (Request Payload) для передачі даних у генератор документів

Графічний інтерфейс вебсервісу спроектовано на основі принципів мінімалізму, що дозволяє оптимізувати швидкість взаємодії оператора з системою та знизити когнітивне навантаження. Користувацький сценарій (User Journey) побудований за лінійною логікою. Робота починається на головній сторінці, де користувач обирає тип цільового документа із запропонованого переліку. Після цього система адаптує екранний простір, відображаючи відповідну інтерактивну форму.

Для заповнення специфікацій, актів чи рахунків у формі передбачено динамічний блок. За допомогою кнопки додавання позицій клієнт може масштабувати табличну частину інтерфейсу – скрипти JavaScript миттєво рендерять нові поля введення без перезавантаження сторінки. Натискання фінальної кнопки генерації запускає асинхронну відправку зібраного JSON-пакета на бекенд за допомогою технології Fetch. На час обробки запиту інтерфейс блокується, а користувачеві демонструється візуальний індикатор завантаження (spinner), який після завершення операції змінюється на пряме посилання для скачування готового файла.

Ергономіка та зручність використання (usability) інтерфейсу забезпечуються декількома базовими механізмами:

- Повна адаптивність (Responsive Design): сітка елементів та форми введення коректно підлаштовуються під будь-яку роздільну здатність екрана, від великих моніторів до дисплеїв портативних планшетів та смартфонів.
- Попередня клієнтська валідація: система автоматично перевіряє заповнення обов'язкових полів до моменту відправки форми на сервер. Якщо виявлено порожні клітинки, інтерфейс підсвічує їх червоним кольором та виводить підказку, що запобігає передачі некоректних даних.
- Швидкодія та зворотний зв'язок: архітектура фронтенду та бекенду оптимізована таким чином, щоб чистий час очікування рендерингу документа в умовах нормального мережевого з'єднання не перевищував 2–3 секунд, що відповідає сучасним стандартам UX для вебзастосунків.

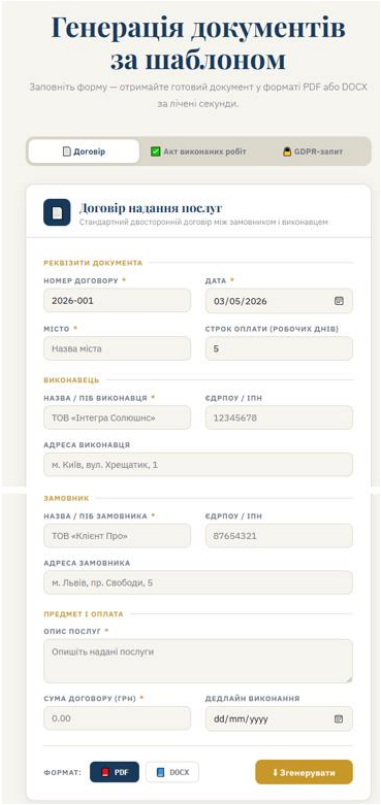


Рис. 2.4 Головна сторінка вебсервісу «DocGen» із формою заповнення реквізитів договору

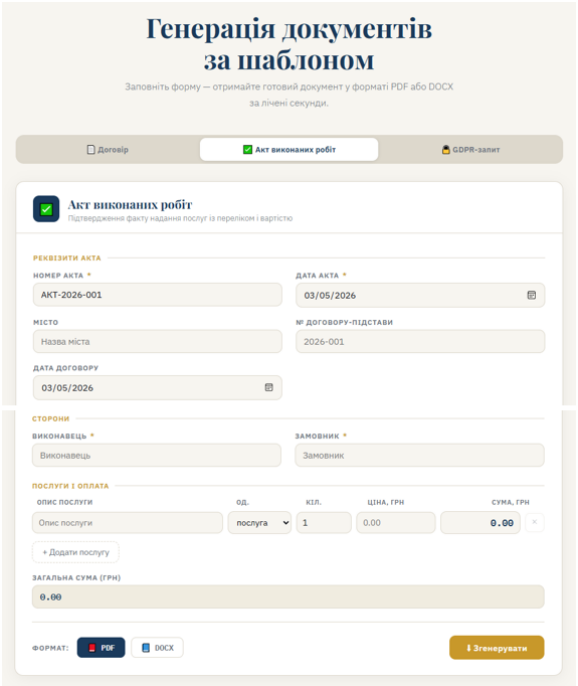


Рис. 2.5 Інтерфейс генерації Акта виконаних робіт із модулем динамічного додавання послуг

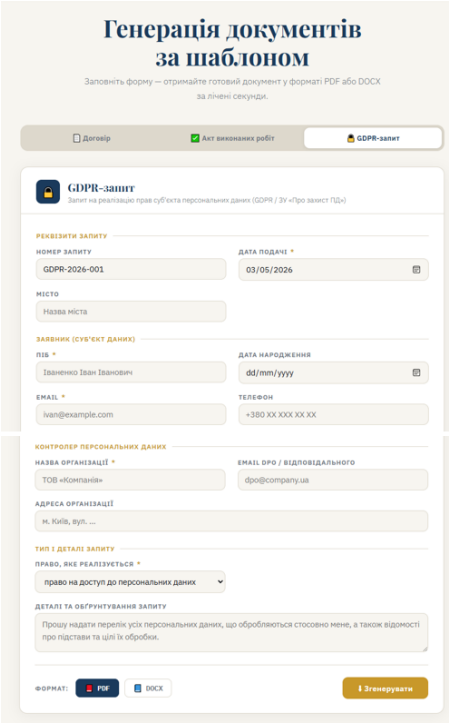


Рис. 2.6 Інтерфейс користувача при роботі з юридичними шаблонами у сфері захисту персональних даних

2.4 Проектування механізмів маніпуляції даними

В основі механізму маніпуляції даними лежить метод ін’єкції динамічного контенту у заздалегідь підготовлені файли-шаблони. Програмна логіка розмежовується на два автономні вектори обробки інформації: підстановку ізольованих текстових констант (поодиноких змінних) та ітераційне розгортання структурованих масивів даних (масивів об'єктів для таблиць).

2.4.1 Семантична розмітка та обробка текстових тегів

Для забезпечення коректної взаємодії серверної частини вебсервісу з візуальним макетом документа розроблено систему умовних маркерів. Синтаксичний аналіз шаблону базується на використанні подвійних фігурних дужок, які обрано як стандартні обмежувачі для змінних. Наявність таких унікальних конструкцій дозволяє

парсеру однозначно диференціювати статичний текст документа від динамічних полів, які підлягають заміні, мінімізуючи ризик помилкового парсингу сусідніх символів.

Проектування алгоритму текстової заміни орієнтоване на збереження вихідного форматування документа. Програмний модуль розроблено так, щоб нове текстове значення, отримане від користувача через API, повністю успадковувало стильові властивості XML-структури конкретного параграфа (run). Сюди відносяться гарнітура шрифту, його розмір (кегль), насиченість, колір та міжрядковий інтервал, задані в оригінальному бланку.

Окремим інженерним завданням є обробка довгих текстових рядків. Механізм заміни враховує поведінку блокових елементів, забезпечуючи автоматичне перенесення слів та пропорційне збільшення висоти комірок або параграфів у разі перевищення стандартного ліміту символів. Це запобігає накладанню тексту на сусідні графічні елементи чи межі таблиць, гарантуючи збереження геометрії та візуальної цілісності фінального документа.

2.4.2 Ітераційне проектування динамічних таблиць

Формування документів із перемінною кількістю позицій (актів приймання-передачі, комерційних рахунків, специфікацій) вимагає реалізації алгоритму динамічного масштабування табличних структур. Процес ітераційної обробки та розширення таблиць на основі вхідного масиву даних містить кілька послідовних кроків.

На першому етапі здійснюється синтаксичний пошук рядка-шаблону. Серверний модуль сканує XML-структуру таблиці в DOCX-файлі, ідентифікуючи рядок, у комірках якого розміщено маркери ітераційних змінних. Цей рядок визначається системою як базовий зразок для подальшого тиражування.

Після фіксації цільового елемента запускається механізм глибокого копіювання (Cloning). Для кожного окремого об'єкта, переданого користувачем у вхідному масиві JSON, програма створює нову ізольовану копію знайденого рядка-зразка. Процедура клонування реалізована на низькому рівні маніпуляції XML-деревом документа, що дозволяє повністю дублювати фізичні властивості оригінальних комірок: товщину та колір меж, внутрішні відступи (padding), вертикальне і горизонтальне вирівнювання, а також параметри фонового затінення.

Створені дублікати інтегруються в тіло таблиці, після чого запускається алгоритм ін'єкції даних. У комірки кожної нової структури послідовно підставляються відповідні динамічні параметри: порядковий номер, найменування виконаних робіт, одиниці виміру, кількісні показники та ціна. Такий підхід дозволяє масштабувати таблицю на довільну кількість рядків, повністю зберігаючи первинну верстку бланка та пропорції сусідніх елементів сторінки.

Паралельно з рендерингом табличної частини архітектура мікросервісу реалізує логіку автоматичних калькуляцій на стороні бекенду. Програма ітераційно прораховує проміжні суми для кожного рядка, а також агрегує фінальні фінансові метрики. Перед фінальним збереженням файла обчислювальний модуль розраховує загальну вартість послуг, суму податкових нарахувань відповідно до обраної системи оподаткування та формує підсумковий текстовий рядок із сумою прописом. Це автоматизує рутинні математичні розрахунки та мінімізує ризик виникнення помилок, зумовлених людським фактором.

2.4.3 Контроль вихідного результату

Заключний етап життєвого циклу маніпуляції даними орієнтований на верифікацію сформованих структур та забезпечення цілісності вихідного файла перед його безпосередньою передачею користувачу. На цьому рівні архітектури вебсервісу

реалізовано двокомпонентний механізм контролю, який поєднує серверу валідацію вхідних масивів та фінальний рендеринг цільових форматів.

Серверна валідація виконує роль захисного бар'єра, що запобігає генерації юридично некоректних або пошкоджених документів. Перед запуском процесу ін'єкції контенту в шаблон обчислювальний модуль здійснює перевірку наявності всіх обов'язкових атрибутів. Якщо під час аналізу JSON-пакета виявляється відсутність критично важливих констант (наприклад, унікального номера акта, дат або ідентифікаційних кодів сторін), алгоритм автоматично блокує подальшу генерацію. У цей момент система ініціює обробку виняткової ситуації (Exception) та повертає на сторону клієнта структуроване повідомлення про помилку з точним зазначенням некоректного поля, що дозволяє користувачеві оперативно виправити дані у формі введення.

Після успішного проходження валідації та завершення підстановки змінних виконується фінальна трансляція документа у компильований файл. Залежно від запиту оператора, система здійснює збирання DOCX-архіву або конвертацію бінарного потоку у формат PDF. Процес рендерингу супроводжується примусовим кодуванням символів у стандарті UTF-8 та динамічним вбудовуванням шрифтових гліфів (font embedding). Це гарантує абсолютну автентичність відображення тексту, повну підтримку специфічних символів українського алфавіту (таких як літери «є», «ї», «ґ» та знака апострофа), а також стабільність геометричних пропорцій і верстки документа на будь-яких пристроях чи операційних системах користувача.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБСЕРВІСУ ГЕНЕРАЦІЇ ДОКУМЕНТІВ

Практичний етап створення спроектованого вебсервісу охоплює написання вихідного коду серверної та клієнтської частин, розробку алгоритмів обробки динамічних масивів, а також інтеграцію компонентів у цілісну програмну систему. Реалізація проєкту орієнтована на створення відмовостійкого застосунку з відкритим синтаксисом і високою швидкістю виконання операцій.

3.1 Опис реалізації програмного забезпечення

Програмне втілення архітектури вебсервісу виконано з використанням мови програмування Python та концепції побудови додатків за патерном REST API. Вибір цього стеку зумовлений високою ефективністю мови під час виконання операцій над текстовими стринговими масивами, а також наявністю розвинених низькорівневих бібліотек, здатних взаємодіяти безпосередньо зі специфікаціями архітектури OpenXML (формат DOCX) та внутрішньою структурою об'єктів PDF.

У межах серверної логіки розроблено серію автономних скриптів, які координують життєвий цикл запиту: від моменту перехоплення JSON-пакета з фронтенду до трансляції згенерованого бінарного потоку назад у браузер користувача. Інженерне рішення відмовитися від традиційних реляційних баз даних на користь обробки даних у реальному часі суттєво спростило архітектуру коду. Це дозволило мінімізувати накладні витрати серверних ресурсів на операції введення-виведення (I/O) та забезпечити високу швидкість рендерингу файлів навіть в умовах обмежених обчислювальних потужностей хмарного хостингу.

3.1.1 Середовище розробки та конфігурація проекту

Інженерне проєктування та написання вихідного коду вебсервісу реалізовано в інтегрованому середовищі (IDE) Visual Studio Code, оснащеному спеціалізованими плагінами для лінтингу (flake8) та форматирування (Black) Python-скриптів. Локальний контроль версій, фіксація змін та керування гілками розробки здійснювалися за допомогою розподіленої системи Git, що дозволило згодом організувати безперебійну синхронізацію з віддаленим репозиторієм на платформі GitHub.

Для запобігання конфліктів між системними компонентами операційної системи та версіями зовнішніх пакетів у проєкті застосовано інструмент venv. Створення ізольованого віртуального середовища дозволило локалізувати всі інстальовані залежності безпосередньо в межах директорії застосунку.

Повний перелік задіяних модулів, фреймворків та службових утиліт із чітким зазначенням їхніх робочих версій зафіксовано в конфігураційному файлі requirements.txt. Такий підхід гарантує точне відтворення архітектури середовища виконання (runtime) та забезпечує ідентичність поведінки вебсервісу як під час локального тестування на машині розробника, так і в процесі автоматичного збирання контейнера на віддалених серверах хмарної платформи Render.

3.1.2 Архітектура бекенд-частини

Програмна архітектура серверної частини вебсервісу базується на використанні мікрофреймворку FastAPI (або Flask), який виступає в ролі координатора мережевої взаємодії та забезпечує маршрутизацію трафіку між клієнтським інтерфейсом і внутрішніми обчислювальними модулями. Під час розробки структури бекенду було застосовано фундаментальний принцип проектування програмного забезпечення – розділення обов'язків (Separation of Concerns). Це дозволило декомпозувати систему

на кілька автономних, слабко пов'язаних між собою функціональних шарів, що значно спрощує подальше тестування та модернізацію окремих компонентів програми.

Першим рівнем цієї архітектури є шар маршрутизації (Routes / Endpoints). Його функціонал обмежений перехопленням вхідних HTTP-запитів від фронтенду, безпосереднім витягуванням сирого JSON-пакета з тіла запиту та передачею цих даних на наступний етап обробки. Маршрутизатори не містять алгоритмів обчислень, а лише визначають точки доступу до API та повертають клієнту фінальний результат або HTTP-статуси помилок.

Проміжною та найбільш інтелектуальною ланкою є шар бізнес-логіки (Services). Саме тут зосереджені ключові алгоритми маніпуляції інформаційними масивами. Модулі цього рівня відповідають за десеріалізацію вхідних даних, запуск процедур комплексної серверної валідації, математичне обчислення фінансових підсумків та структурну підготовку масивів для передачі в динамічні таблиці. Шар бізнес-логіки повністю ізольований від мережевих протоколів, що дозволяє перевикористовувати його функції в інших компонентах системи.

За безпосереднє формування документів відповідає шар шаблонізації (Templates / Generators). Він взаємодіє з файловою системою сервера, де зберігаються заздалегідь підготовлені бланки документів із семантичною розміткою. Отримуючи очищені й структуровані дані від шару бізнес-логіки, цей модуль ініціює низькорівневі процеси підстановки тегів, тиражування рядків та фінальної компіляції вихідного бінарного файлового потоку.

3.1.3 Механізм обробки документів та маніпуляція даними

Функціональний алгоритм генерації вихідних файлів базується на концепції динамічного інжекту контенту, який адаптується під архітектурні особливості цільових форматів DOCX та PDF. Програмна логіка абстрагована від статичного заповнення полів і працює безпосередньо з об'єктними моделями документів.

Під час опрацювання файлів формату DOCX серверний модуль завантажує оригінальний файл-шаблон у пам'ять як структуроване XML-дерево. За допомогою інструментів бібліотеки python-docx програма здійснює поблочний обхід документа. Алгоритм виконує ітерацію по колекціях параграфів (doc.paragraphs) та текстових запусків (runs), аналізуючи їхній вміст на наявність регулярних виразів, що відповідають масці тегів. Процес заміни маркерів на реальні значення реалізовано на рівні об'єктів Run. Це дозволяє вносити зміни безпосередньо в текстові вузли, не зачіпаючи сусідні XML-теги стилізації, завдяки чому повністю зберігається початкове форматування (колір, розмір та шрифт).

Особливе місце в архітектурі цього модуля займає алгоритм динамічного розширення таблиць (Table DOM). Замість базового виклику методу table.add_row(), який у стандартній реалізації створює порожній рядок без збереження візуальних властивостей, у проєкті розроблено функцію низькорівневого копіювання елементів trPr (Table Row Properties) та tcPr (Table Cell Properties). Програма дублює XML-параметри рядка-зразка, переносить їх на новий ітераційний рядок і лише після цього заповнює комірки текстовими даними з масиву Items. Таке інженерне рішення гарантує стабільність сітки таблиці, константність меж та внутрішніх відступів незалежно від кількості згенерованих позицій.

Обробка та експорт документів у форматі PDF виділені в окремий незалежний підмодуль на базі бібліотеки ReportLab. Зважаючи на те, що PDF є специфікацією з фіксованою розміткою (векторним контейнером), логіка генерації побудована на використанні концепту двовимірної координатної сітки та об'єктів керування потоком Platypus (Page Layout and Typography Using Scripts). Програмний модуль динамічно розраховує висоту текстових блоків, використовуючи класи Paragraph та Spacer, і автоматично контролює розбиття контенту на сторінки (Page Breaking). Для забезпечення гнучкості інтерфейсу, у випадках, коли обсяг внесеного тексту перевищує стандартні межі, алгоритм автоматично зміщує Y-координату наступних

елементів, запобігаючи накладанню шарів та виходу інформації за межі друкованої області (скропінгу).

3.1.4 Вирішення проблеми локалізації та кодування

Під час розробки систем автоматичного формування текстового контенту в україномовному сегменті критичним завданням є забезпечення коректної локалізації та підтримки кодування символів. Стандартні модулі генерації PDF-файлів у багатьох базових бібліотеках Python за замовчуванням орієнтовані на західноєвропейські кодові сторінки (наприклад, Latin-1) та використовують обмежений набір вбудованих шрифтів (Helvetica, Times-Roman, Courier), які не містять кириличних гліфів. Це призводить до виникнення критичних помилок під час рендерингу, коли замість специфічних українських літер («і», «ї», «є», «ґ», «в») у фінальному документі відображаються порожні прямокутники або некоректні символи (ефект «кракозябр»).

Для подолання цієї проблеми на рівні архітектури бекенду було реалізовано механізм примусової локалізації текстових потоків через кодування UTF-8 та динамічну реєстрацію зовнішніх шрифтових гарнітур. Як базовий інструмент інтегровано сімейство шрифтів з відкритим вихідним кодом *DejaVu Sans* (або *TTF-версії* загальноприйнятих системних шрифтів), які мають повне покриття кириличного діапазону Unicode.

Процедура підключення та ініціалізації шрифтових пакетів виконується автоматично на етапі запуску (startup event) сервера FastAPI / Flask і складається з декількох послідовних кроків:

1. За допомогою системного модуля os програма верифікує наявність .ttf файлів (наприклад, DejaVuSans.ttf та його напівжирного накреслення DejaVuSans-Bold.ttf) у локальній директорії проєкту assets/fonts/.

2. Використовується низькорівневий модуль pdfmetrics з бібліотеки ReportLab, який викликає метод pdfmetrics.registerFont(). Цей метод зчитує метрики

зовнішнього TrueType-файла, аналізує таблицю гліфів та реєструє нову гарнітуру в глобальному контексті генератора.

3. За допомогою класу TTFont створюється зв'язок між логічним ім'ям шрифту в коді програми та його фізичним файловим представленням.

Після завершення ініціалізації зареєстровані назви шрифтів передаються до шару шаблонізації. Під час рендерингу елементів Paragraph або маніпуляцій з об'єктами canvas.drawString() обчислювальний модуль використовує виключно валідовану назву гарнітури. Такий алгоритм повністю усуває ризик спотворення інформації, гарантує безпомилковий експорт українського тексту і забезпечує ідентичність візуального відображення символів незалежно від платформи, на якій розгорнуто вебсервіс.

3.1.5 Розгортання та CI/CD

Для переведення розробленого вебсервісу у статус релізу та забезпечення його доступності для кінцевих користувачів було організовано процес безперервної інтеграції та розгортання (CI/CD). Як цільове хмарне середовище обрано платформу Render, яка дозволяє автоматизувати життєвий цикл SaaS-додатків без необхідності складного адміністрування фізичних серверних потужностей.

В основі розробленого конвеєра деплою лежить подійно-орієнтована архітектура, реалізована за допомогою механізму вебхуків (Webhooks) та прямої інтеграції хмарного сервісу з віддаленим репозиторієм проєкту на GitHub. Автоматизований цикл публікації оновлень містить кілька послідовних стадій:

1. Фіксація змін (Trigger): розробник виконує команду git push, відправляючи стабільну версію вихідного коду або виправлених шаблонів документів у головну гілку (main / master) віддаленого репозиторію.

2. Ініціалізація збирання (Build Stage): Платформа Render перехоплює подію оновлення коду через вебхук і миттєво запускає ізольоване середовище збирання (build container).

3. Управління залежностями: На етапі компіляції хмарний сервер аналізує файл requirements.txt. Система автоматично завантажує та інсталює всі необхідні бібліотеки, включаючи fastapi, python-docx, reportlab та pandas, фіксуючи їх у внутрішньому кеші для прискорення наступних деплоїв.

4. Конфігурація оточення (Environment): Вебсервер зчитує змінні оточення (Environment Variables), які зберігаються в зашифрованому вигляді на стороні Render. Це дозволяє безпечно керувати глобальними налаштуваннями додатку без їхнього явного прописування у вихідному коді.

5. Запуск та моніторинг (Deployment & Health Check): Після успішного завершення етапу збирання Render запускає ASGI-сервер uvicorn (або gunicorn). Нова версія застосунку починає обробляти трафік лише після проходження внутрішнього тесту на працездатність (Health Check). Стара версія сервера відключається автоматично і безперервно, що гарантує нульовий час простою застосунку (Zero-Downtime Deployment) для користувачів.

Така схема автоматизації дозволила повністю виключити рутинні ручні операції під час оновлення продукту, звела до мінімуму ймовірність виникнення помилок конфігурації сервера та забезпечила швидке й безпечне тестування нових функцій вебсервісу в реальному часі.

3.2 Специфікація ключових класів та методів

Програмна архітектура серверного компонента вебсервісу реалізована із суворим дотриманням парадигми об'єктно-орієнтованого програмування (ООП). Таке інженерне рішення дозволило інкапсулювати низькорівневі механізми взаємодії з

файловими структурами OpenXML та PDF-потоками всередині ізольованих класів-сервісів. Завдяки декомпозиції логіки розроблений код відповідає критеріям масштабованості та легкості супроводу: у разі необхідності інтеграції нового типу бланка (наприклад, специфічної кадрової звітності чи внутрішніх розпоряджень) розробнику достатньо розширити наявну ієрархію класів або додати новий метод обробки, не вносячи змін до конвеєра маршрутизації та логіки роботи вебсервера.

Центральним обчислювальним вузлом системи є спеціалізований модуль-генератор. Він функціонує як проміжний адаптер (модуль бізнес-логіки), який трансформує десеріалізовані з JSON-пакета структури даних у фізичні параметри маніпуляції шаблонами. Вся логіка генерації розподілена між двома ключовими класами, кожен з яких відповідає за свій тип вихідного файлового контейнера.

Перший базовий клас оперує методами обробки текстових файлів. До його обов'язків входить ініціалізація об'єкта документа, відкриття файлового дескриптора шаблону, а також координація функцій для поблочної заміни текстових маркерів та клонування елементів таблиць на основі ітераційних масивів. Другий клас інкапсулює в собі логіку формування векторних PDF-документів за допомогою методів точного позиціонування об'єктів на двовимірній сітці сторінки, управління відступами та динамічного підрахунку висоти контенту для автоматичного створення нових сторінок. Керування цими процесами з єдиної точки доступу спрощує контроль виняткових ситуацій (Exception Handling) та гарантує стабільність вихідних результатів.

Таблиця 3.1

Опис ключових функцій та методів системи

Назва методу	Опис функціоналу
generate_document()	Основний метод контролера. Приймає вхідний JSON-пакет та викликає відповідний шаблонізатор для створення документа.
process_table_rows()	Метод для циклічної обробки даних. Використовується для динамічного додавання нових рядків у таблиці шаблону.
setup_fonts()	Налаштування та реєстрація шрифтів TrueType. Необхідний для коректного відображення кирилиці при конвертації в PDF.
download_file()	Ендпоінт, який відповідає за передачу згенерованого бінарного файлу (стримування даних) на сторону клієнта.

3.2.1 Поглиблений аналіз програмної логіки методів

Детальний аналіз внутрішньої архітектури вебсервісу вимагає ретельного розгляду програмної логіки, інкапсульованої в ключових методах серверної частини. Кожен метод розроблено як ізольований функціональний блок, що відповідає за конкретний етап обробки інформаційних потоків та взаємодії з файловими структурами.

Метод generate_document(data, template_type)

Цей метод є головною вхідною точкою для запуску процесів бізнес-логіки на бекенді. Функція приймає два аргументи: словник користувацьких параметрів data та ідентифікатор цільового бланка template_type. Логіка виконання методу розпочинається з блоку валідації – програма перевіряє наявність обов'язкових ключів у структурі отриманого словника.

У разі успішного проходження верифікації метод звертається до дискового простору сервера, визначає абсолютний шлях до необхідного файлу-шаблону та ініціалізує його об'єктну модель (DOM). Далі запускається внутрішній парсер, який

здійснює пошук семантичних маркерів у тексті та замінює їх на відповідні стрингові значення. Після завершення підстановок метод конвертує результуючий бінарний потік у буфер оперативної пам'яті (io.BytesIO) або створює тимчасовий зашифрований файл у робочій директорії Runtime Storage на сервері Render, повертаючи посилання на створений ресурс.

Метод process_table_rows(table_object, items_list)

Цей метод вирішує завдання динамічного масштабування таблиць OpenXML, долаючи статичні обмеження стандартних DOCX-файлів. Як параметри функція приймає об'єкт таблиці table_object та масив об'єктів items_list, кожен елемент якого містить відомості про окрему позицію товару чи послуги.

Алгоритм роботи методу побудований за наступною схемою. Спочатку програма сканує таблицю і фіксує передостанній рядок, який містить маркери ітераційних змінних і визначений як рядок-зразок. Далі запускається цикл for, який ітериує по кожному елементу списку items_list. На кожному кроці циклу метод виконує низькорівневе глибоке клонування структури рядка-зразка, копіюючи всі його XML-атрибути (параметри шрифтів, колірні схеми, контури меж та внутрішні відступи комірок).

У створені нові комірки послідовно записуються відповідні динамічні дані: текстовий опис позиції, числові показники кількості та ціни за одиницю. Паралельно метод виконує математичне обчислення проміжної вартості за поточним рядком і додає отримане значення до накопичувальної змінної загального підсумку. Після завершення циклу вихідний рядок-зразок видаляється з XML-дерева, що забезпечує коректний вигляд готової таблиці без порожніх технічних елементів.

Метод setup_fonts()

Цей метод виконує конфігураційну функцію та є базою для коректної локалізації інтерфейсу й вихідних документів. Метод не приймає зовнішніх аргументів і викликається автоматично під час ініціалізації вебсервера.

Основна задача функції – усунути обмеження базових бібліотек PDF-генерації щодо роботи з кодуванням UTF-8. Метод setup_fonts() звертається до папки з ресурсами проєкту, зчитує TrueType-файли кириличних шрифтів (наприклад, DejaVuSans.ttf) та реєструє їх у глобальному реєстрі pdfmetrics. Завдяки цьому під час формування PDF-файлів обчислювальний модуль ReportLab отримує доступ до повного набору гліфів українського алфавіту, що повністю запобігає спотворенню тексту або появі порожніх символів у фінальних звітах.

Метод download_file(file_path)

Реалізація цього методу відповідає за безпечну та оптимізовану передачу згенерованого контенту на сторону клієнта. Замість повного завантаження файла в оперативну пам'ять сервера перед відправкою, метод здійснює поблокове зчитування та трансляцію даних через бінарний потік (streaming response). Це інженерне рішення мінімізує навантаження на RAM-пам'ять сервера Render під час пікових навантажень або одночасної роботи багатьох користувачів.

Додатково метод конфігурує HTTP-заголовки відповіді, встановлюючи параметр Content-Disposition у значення attachment із зазначенням імені файлу. Такий HTTP-метазапис є прямою інструкцією для браузера користувача: він блокує спроби відкрити документ у поточній вкладці та миттєво ініціює скачування файлу на локальний пристрій через стандартний менеджер завантажень.

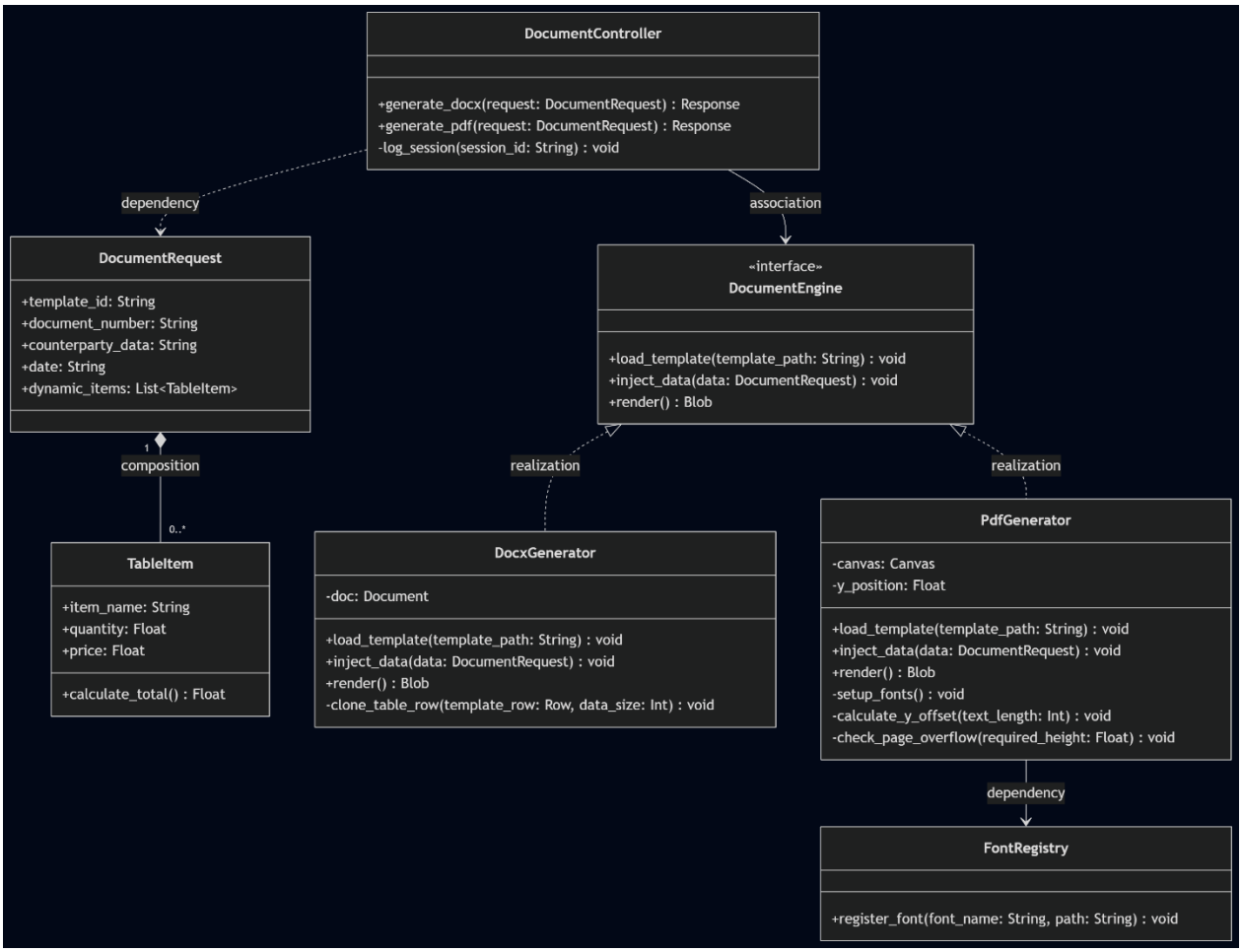


Рис. 3.1 Діаграма класів (Class Diagram)

3.3 Алгоритм динамічної маніпуляції таблицями

Однією з найбільш трудомістких задач при розробці сервісу стала реалізація підтримки багаторядкових табличних структур у документах форматів DOCX та PDF. На відміну від статичних текстових полів, таблиці вимагають динамічної зміни структури файлу без порушення загальної цілісності верстки та стилістики документа.

3.3.1 Логіка роботи з об'єктами таблиць

Процес маніпуляції даними в табличних структурах базових шаблонів спирається на комплексний інженерний алгоритм ітеративного клонування та динамічного заповнення. Розроблений програмний модуль здійснює безпосереднє низькорівневе керування внутрішньою XML-структурою документа під час обробки формату DOCX, а в разі генерації PDF-файлів оперує об'єктами координат і логікою керування текстовими потоками.

Алгоритм починає роботу з ідентифікації цільового об’єкта в процесі глибокого парсингу внутрішнього дерева файла. Програма сканує колекцію таблиць і знаходить потрібну структуру за її позиційним індексом або за унікальним текстовим маркером. Після локалізації таблиці система аналізує еталонне форматування для збереження оригінального стилю бланка. Скрипт зчитує та фіксує в оперативній пам'яті всі властивості останнього робочого рядка, включаючи гарнітуру шрифтів, колірні схеми заповнення комірок та конфігурацію їхніх меж. Цей рядок надалі виконує роль технічного еталона для всього масиву нових даних.

Наступна фаза алгоритму полягає в ітераційній генерації рядків, де програма запускає цикл обробки масиву, отриманого від клієнта через вебформу застосунку. Кількість ітерацій циклу чітко відповідає кількості об’єктів у надісланому JSON-пакеті. На кожному кроці розроблений метод створює новий фізичний рядок у поточній таблиці, копіюючи збережені XML-властивості еталона.

Безпосередньо під час ітерації виконується ін’єкція даних та їхнє контекстне форматування. Алгоритм послідовно звертається до конкретних комірок новоствореного рядка, затирає технічні маркери та записує фактичні значення, такі як найменування наданої послуги, її кількісні показники та ціну за одиницю. При цьому нові текстові та числові вузли повністю успадковують стилістичні параметри оригінального осередку.

Завершується робота алгоритму математичним розрахунком та повною фіналізацією структури документа. Після успішного заповнення всіх динамічних рядків обчислювальний модуль бекенду в автоматичному режимі калькулює підсумкові фінансові показники. Програма підсумовує вартість усіх позицій, вираховує податкові нарахування та вносить отримані агреговані значення у фінальний підсумковий рядок таблиці. Такий підхід гарантує математичну точність обчислень, запобігає руйнуванню геометричних пропорцій документа та забезпечує високу якість фінального візуального результату.

3.3.2 Програмна реалізація алгоритму на мові Python

Нижче наведено фрагмент коду, що демонструє практичне застосування описаного алгоритму за допомогою бібліотеки python-docx:

```
# ---------------------------------------------------------------------------
# PDF GENERATION
# ---------------------------------------------------------------------------

def generate_pdf(doc_data):
    buffer = io.BytesIO()
    doc = SimpleDocTemplate(
        buffer, pagesize=A4,
        rightMargin=2*cm, leftMargin=2.5*cm,
        topMargin=2*cm, bottomMargin=2*cm,
    )
    styles = getSampleStyleSheet()

    title_style = ParagraphStyle("DocTitle", parent=styles["Normal"],
        fontSize=14, leading=18, alignment=TA_CENTER,
        spaceAfter=4, fontName=PDF_FONT_BOLD)
    subtitle_style = ParagraphStyle("DocSubtitle", parent=styles["Normal"],
```

```
    fontSize=9, leading=12, alignment=TA_CENTER, spaceAfter=2,
    fontName=PDF_FONT_ITALIC, textColor=colors.HexColor("#555555"))
meta_style = ParagraphStyle("DocMeta", parent=styles["Normal"],
    fontSize=10, leading=14, alignment=TA_CENTER,
    spaceAfter=6, fontName=PDF_FONT)
heading_style = ParagraphStyle("SectionHeading", parent=styles["Normal"],
    fontSize=11, leading=14, spaceBefore=14, spaceAfter=4,
    fontName=PDF_FONT_BOLD, textColor=colors.HexColor("#1a3a5c"))
body_style = ParagraphStyle("BodyText", parent=styles["Normal"],
    fontSize=10, leading=15, alignment=TA_JUSTIFY,
    spaceAfter=4, fontName=PDF_FONT)
sign_style = ParagraphStyle("SignLine", parent=styles["Normal"],
    fontSize=10, leading=20, fontName=PDF_FONT)

story = []
story.append(Spacer(1, 0.3*cm))
story.append(Paragraph(doc_data["title"], title_style))
if doc_data.get("subtitle"):
    story.append(Paragraph(doc_data["subtitle"], subtitle_style))
story.append(Paragraph(
    "№ %s  |  %s  |  %s" % (doc_data["number"], doc_data["city"], doc_data["date"]),
    meta_style))
story.append(HRFlowable(width="100%", thickness=1.5, color=colors.HexColor("#1a3a5c")))
story.append(Spacer(1, 0.4*cm))

for section in doc_data["sections"]:
    story.append(Paragraph(section["heading"], heading_style))
    if "body" in section:
        for line in section["body"].split("\n"):
            story.append(Paragraph(line or " ", body_style))
    if "table" in section:
        t = section["table"]
```

```
            table_data = [t["headers"]] + t["rows"] + [["", "", "", "", "РАЗОМ:", t["total"]]]
            col_widths = [1*cm, 7*cm, 2*cm, 2*cm, 2.5*cm, 2.5*cm]
            tbl = Table(table_data, colWidths=col_widths)
            tbl.setStyle(TableStyle([
              ("FONTNAME",      (0,0),  (-1,-1), PDF_FONT),
              ("BACKGROUND",    (0,0),  (-1,0),  colors.HexColor("#1a3a5c")),
              ("TEXTCOLOR",     (0,0),  (-1,0),  colors.white),
              ("FONTNAME",      (0,0),  (-1,0),  PDF_FONT_BOLD),
              ("FONTSIZE",      (0,0),  (-1,-1), 9),
              ("ALIGN",         (0,0),  (-1,-1), "CENTER"),
              ("ALIGN",         (1,1),  (1,-2),  "LEFT"),
              ("GRID",          (0,0),  (-1,-2), 0.5, colors.HexColor("#cccccc")),
              ("BACKGROUND",    (0,-1), (-1,-1), colors.HexColor("#e8f0fe")),
              ("FONTNAME",      (0,-1), (-1,-1), PDF_FONT_BOLD),
              ("LINEABOVE",     (0,-1), (-1,-1), 1, colors.HexColor("#1a3a5c")),
              ("ROWBACKGROUNDS",(0,1),  (-1,-2), [colors.white, colors.HexColor("#f5f8ff")]),
              ("TOPPADDING",    (0,0),  (-1,-1), 5),
              ("BOTTOMPADDING", (0,0),  (-1,-1), 5),
            ]))
            story.append(tbl)
            story.append(Spacer(1, 0.3*cm))

        story.append(Spacer(1, 0.8*cm))
        story.append(HRFlowable(width="100%", thickness=0.5, color=colors.HexColor("#aaaaaa")))
        story.append(Spacer(1, 0.4*cm))
        sign_data = [
          [Paragraph("ВИКОНАВЕЦЬ / ЗАЯВНИК", sign_style), Paragraph("ЗАМОВНИК / ОТРИМУВАЧ", sign_style)],
          [Paragraph("_____ /_____/", sign_style), Paragraph("_____ /_____/", sign_style)],
        ]
        sign_table = Table(sign_data, colWidths=[9*cm, 9*cm])
```

```
sign_table.setStyle(TableStyle([
   ("ALIGN",      (0,0), (-1,-1), "LEFT"),
   ("FONTSIZE",   (0,0), (-1,-1), 10),
   ("TOPPADDING", (0,0), (-1,-1), 6),
]))
story.append(sign_table)
doc.build(story)
buffer.seek(0)
return buffer.read()
```

3.3.3 Особливості маніпуляції даними для PDF-формату

Під час генерації вихідних документів у форматі PDF розроблений алгоритм має специфічні інженерні особливості, зумовлені фіксованою геометричною природою цього файлового контейнера. На відміну від форматів текстових процесорів, PDF-документ є набором статичних векторних інструкцій і не підтримує автоматичне динамічне переверстування або розтягування табличних комірок. Через це в межах програмного підмодуля на базі бібліотеки ReportLab було реалізовано комплекс низькорівневих обчислювальних механізмів для точного позиціонування контенту.

Основою цього механізму є алгоритм динамічного розрахунку двовимірних координат. Система в автоматичному режимі покроково обчислює точне зміщення кожної наступної лінії тексту та елементів таблиці по вертикальній осі Y, де початком координат традиційно є лівий нижній кут сторінки. Програма аналізує висоту шрифту та міжрядкові інтервали (leading), що дозволяє превентивно уникати накладання текстових шарів один на одного під час фінального рендерингу.

Поряд із цим критично важливим елементом логіки є постійний контроль перенесення даних на нову сторінку (Page Breaking). Перед ін'єкцією кожного наступного рядка таблиці з масиву даних алгоритм ретельно перевіряє залишкову висоту поточної робочої області аркуша. Якщо координата Y наближається до

критичної межі нижнього поля (margin), обчислювальний модуль блокує подальший рендеринг на поточному листі.

Програма автоматично ініціює створення нової сторінки за допомогою виклику методу canvas.showPage(), скидає значення поточної координати Y до верхньої робочої межі та примусово здійснює повторне дублювання рядка заголовків («шапки») таблиці. Такий підхід забезпечує читабельність багатосторінкових документів, зберігає сувору структуру звітів та повністю виключає ризик втрати інформації або її виходу за межі друкованої області аркуша.

Рис. 3.3 Flowchart (блок-схема) алгоритму

3.4 Опис функціонування та інтерфейсу користувача

Функціонування вебсервісу базується на принципах інтерактивності, безперервного зворотного зв'язку та мінімізації дій оператора (User Actions). Графічний інтерфейс користувача спроектовано з урахуванням сучасних вимог до

ергономіки, що дозволяє оптимізувати процес маніпуляції складними структурованими даними та прискорити генерацію фінальних файлів.

3.4.1 Логіка взаємодії компонентів (Граф діалогів)

Взаємодія користувача з інтерфейсом системи є лінійним процесом, побудованим на послідовному виконанні взаємопов'язаних операцій, які утворюють цілісний цикл обробки цифрової інформації. Робоча сесія розпочинається на головному екрані застосунку, де користувачеві демонструється інтерактивна панель керування. Тут відбувається ініціалізація сесії та вибір типу документа. Залежно від обраного пункту (наприклад, акт виконаних робіт, договір чи кадровий запит), фронтенд-модуль динамічно підвантажує відповідну структуру вебформи, специфічну для конкретного бізнес-сценарію.

Після рендерингу цільової форми розпочинається введення статичних атрибутів документа. Користувач заповнює константні поля, що формують заголовок і підвал майбутнього бланка: унікальний номер, календарну дату, назву організації та юридичні реквізити контрагента. На цьому етапі підключаються інтегровані JavaScript-валідатори, які в реальному часі здійснюють превентивну перевірку коректності введення даних, аналізують формат дат та контролюють заповненість обов'язкових полів для запобігання надсиланню порожніх запитів.

Центральною частиною інтерфейсної логіки є механізм динамічного керування табличними масивами. За допомогою спеціалізованої кнопки додавання позицій реалізовано алгоритм маніпуляції деревом DOM (Document Object Model). Кожне клікання по цій кнопці активує скрипт, який миттєво генерує новий ізольований набір полів для опису товарів чи послуг прямо всередині поточної форми. Це дозволяє оператору гнучко масштабувати табличну частину, додавати необмежену кількість рядків або видаляти помилково внесені позиції без перезавантаження сторінки.

Після завершення підготовки даних запускається процес асинхронного обміну інформацією з сервером. Натискання фінальної кнопки генерації перехоплюється встановленим слухачем подій (Event Listener). Система автоматично зчитує вміст усіх полів форми, серіалізує інформацію у формат JSON і за допомогою інструменту Fetch API відправляє POST-запит на бекенд-ендпоінт. Для підвищення рівня зручності (UX) на час обробки запиту інтерфейс сторінки частково блокується, а користувачеві демонструється анімований індикатор завантаження.

Цикл взаємодії завершується автоматичним отриманням результату. Сервер обробляє запит, виконує рендеринг і повертає бінарний потік даних. Клієнтський сценарій перехоплює цю відповідь, трансформує бінарний потік у спеціалізований об'єкт Blob (Binary Large Object), динамічно створює тимчасове посилання в пам'яті браузера та ініціює скачування готового документа на локальний пристрій через стандартний менеджер завантажень.

3.4.2 Нюанси програмної реалізації інтерфейсу

Під час розробки користувацького інтерфейсу та його інтеграції з серверною частиною було враховано низку технічних нюансів, пов'язаних із синхронізацією даних та оптимізацією клієнтського досвіду (UX). Основна інженерна задача полягала в тому, щоб перенести частину обчислювального навантаження на сторону браузера, забезпечивши миттєвий відгук інтерфейсу на дії користувача.

Одним із таких рішень є механізм калькуляції фінансових показників у режимі реального часу (Real-Time Calculation). Попри те, що фінальну валідацію та юридично точні розрахунки виконує виключно серверний модуль, для зручності оператора в інтерфейс форми вбудовано JavaScript-скрипт. Цей сценарій перехоплює події введення чисел (input) у полях кількості та ціни динамічних рядків. Він миттєво перемножує значення всередині кожного рядка, виводить проміжну суму позиції та агрегує загальний підсумок форми ще до моменту відправки JSON-пакета на сервер.

Такий підхід дозволяє користувачеві візуально контролювати коректність внесених даних та оперативно виявляти друкарські помилки.

Окрему увагу приділено динамічній адаптації форми під структуру обраного шаблону. Оскільки архітектура вебсервісу спроектована з урахуванням концепції «гнучкої таблиці», інтерфейс фронтенду повністю нівелює проблему статичних обмежень, яка часто виникає при ручному заповненні бланкових документів. При додаванні користувачем нових товарних позицій через інтерфейс, клієнтський скрипт не просто дублює поля введення, а автоматично присвоює їм унікальні індекси масиву в межах структури DOM.

Це дозволяє бекенд-модулю під час десеріалізації чітко диференціювати кожен рядок, коректно прораховувати геометричні параметри таблиці OpenXML або PDF-координат та масштабувати документ строго вниз. У результаті кінцевий користувач отримує інструмент для формування актів, специфікацій або договорів на довільну кількість позицій, будучи впевненим, що фінальна верстка, корпоративні шрифти, відступи та пропорції документа залишаться незмінними.

3.4.3 Тестування користувацького досвіду

На заключному етапі розробки було проведено комплексне тестування користувацького досвіду (Usability Testing) та оцінку функціональної ефективності впровадженого вебсервісу. Головною метою випробувань став порівняльний аналіз часових витрат і частоти виникнення помилок під час оперативного формування звітних документів ручним способом та за допомогою розробленого автоматизованого рішення.

Емпіричні дані, отримані під час хронометражу дій операторів, підтвердили високу ефективність створеного інтерфейсу. Середній час, необхідний для формування та повної фіналізації типового акта приймання-передачі виконаних робіт із багаторядковою табличною частиною, скоротився з 10–15 хвилин (при

традиційному ручному заповненні та редагуванні файлів у текстовому процесорі Microsoft Word) до 1–2 хвилин у вікні браузера. Оптимізація часових метрик досягнута завдяки автоматичному підвантаженню структури полів під конкретний шаблон та усуненню рутинних операцій із ручного вирівнювання верстки.

Окрему увагу під час тестування було приділено аналізу надійності даних. Інтеграція алгоритмів автоматичного підрахунку фінансових метрик на стороні клієнта та бекенду, а також впровадження механізмів ітераційного клонування рядків повністю нівелювали ризик виникнення арифметичних помилок. Система заблокувала прояв людського фактора (друкарських помилок, некоректного підсумовування або неправильного розрахунку податкових ставок), що часто трапляється при ручному веденні специфікацій. Результати тестування доводять, що спроектована логіка взаємодії компонентів забезпечує високу швидкість обробки інформації, точність вихідних результатів і є повністю готовою до експлуатації в реальних бізнес-процесах.

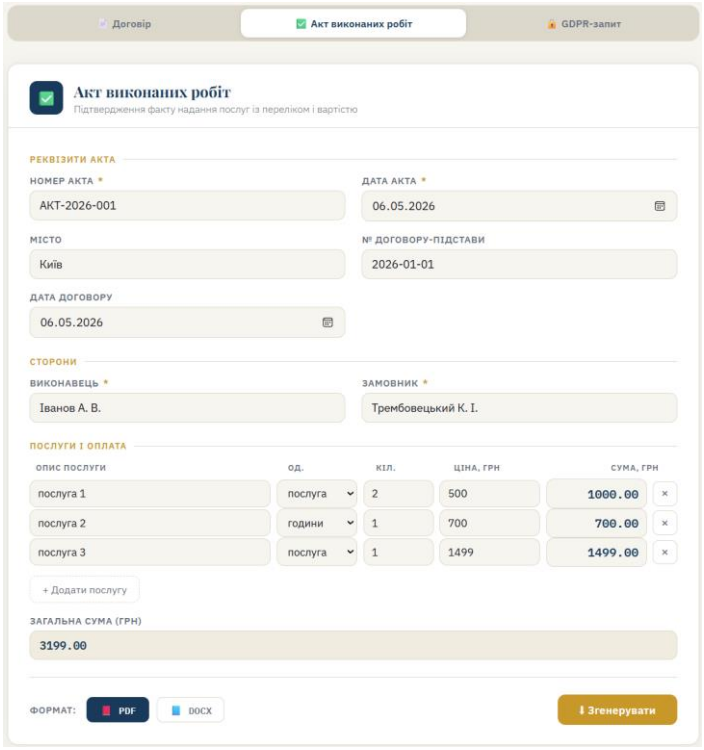


Рис. 3.4 Тестування функціоналу динамічного додавання позицій та коректності автоматичних обчислень у веб-інтерфейсі

АКТ ВИКОНАНИХ РОБІТ

(Надання послуг)

№ АКТ-2026-001 | Київ | 2026-05-06

ЗАГАЛЬНА ІНФОРМАЦІЯ

Виконавець: Іванов А. В.

Замовник: Трембовецький К. І.

Підстава: Договір № 2026-01-01 від 2026-05-06

ПЕРЕЛІК НАДАНИХ ПОСЛУГ

№	Найменування послуги	Од.	Кіл.	Ціна, грн	Сума, грн
1	послуга 1	послуга	2	500.00	1000.00
2	послуга 2	години	1	700.00	700.00
3	послуга 3	послуга	1	1499.00	1499.00
				РАЗОМ:	**3199.00**

ПІДСУМОК

Загальна вартість наданих послуг: 3199.00 грн.

Послуги виконано в повному обсязі та у встановлені строки. Претензій щодо якості та строків надання послуг Замовник не має.

ВИКОНАВЕЦЬ / ЗАЯВНИК

_____ /_____/

ЗАМОВНИК / ОТРИМУВАЧ

_____ /_____/

Рис. 3.5 Зовнішній вигляд вихідного документа «Акт виконаних робіт» у форматі PDF, сформованого автоматично

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метою тестування розробленого веб-сервісу є перевірка відповідності програмног продукту функціональним і технічним вимогам, виявлення та усунення дефектів у логіці обробки даних, а також підтвердження стабільності системи під час роботи в хмарному середовищі Render.

4.1 Обґрунтування вибору видів тестування та стратегії контролю якості

Контроль якості розробленого вебсервісу для автоматичної генерації звітів та документів за шаблоном є фундаментальним етапом інженерного циклу, спрямованим на перевірку відповідності системи заявленим функціональним та нефункціональним вимогам, а також детермінацію її стійкості до зовнішніх експлуатаційних деструкцій. Оскільки архітектура застосунку базується на динамічній обробці XML-структур документів (формат DOCX) та їх трансляцій у графічні контенти (формат PDF), процес верифікації має охоплювати не лише стандартні перевірки логічних розгалужень, але й глибокий аналіз коректності рендерингу та поведінки системи в умовах обчислювального дефіциту.

Стратегія контролю якості передбачає комбінування методів «чорної скриньки» (Black Box Testing) для перевірки відповідності інтерфейсу та кінцевого результату очікуванням користувача, а також методів «білої скриньки» (White Box Testing) на рівні внутрішньої бізем-логіки обробника даних. Основний акцент зміщено в бік автоматизованого тестування критичних модулів парсингу шаблонів та ручного функціонального тестування графічного користувацького інтерфейсу. Реалізація комплексної стратегії дозволяє мінімізувати ризики виникнення архітектурних збоїв на етапі експлуатації вебресурсу в хмарному середовищі.

4.1.1 Інтеграційне тестування (Integration Testing)

Інтеграційне тестування спрямоване на виявлення дефектів під час взаємодії між ізольованими модулями вебзастосунку. Архітектура системи передбачає послідовне передавання даних від клієнтського інтерфейсу користувача до серверної логіки, де відбувається валідація, взаємодія із шаблонізатором та пдальший експорт готових результатів.

У межах інтеграційної верифікації перевіряється коректність роботи інтерфейсу пердачі даних між компонентом зчитування HTTP-запитів і внутрішньою підсистемою генерації документів. Особлива увага приділяється точкам спряження модулів, де вхідні JSON-масиви перетворюються на структурні змінні для підстановки у теги-заповнювачі шаблону DOCX. Окремо тестується інтеграційний стик між логікою генерації вихідного файлу OpenXML та модулем конвертації у PDF-потік. Збої у зазначених вузлах взаємодії призводять до спотворення даних або аварійного завершення робочого процесу, тому тестування фокусується на передачі валідних та інтегрованих об’єктів даних через міжмодульні інтерфейси програмного забезпечення.

4.1.2 Тестування локалізації та рендерингу (Localization Testing)

Важливим етапом перевірки розробленого веб-сервісу стало тестування локалізації та правильності візуального відображення. Окремий аналіз цих параметрів був обумовлений архітектурними особливостями формування PDF-документів та необхідністю усунення ризиків, пов’язаних із кодуванням кирилиці. Процедура перевірки складалася з двох етапів: перевірки сумісності текстових даних із кодуванням UTF-8 та аналізу стабільності вбудовування шрифтів TrueType.

Перший етап передбачав перевірку відображення окремих символів української абетки. Тестові сценарії передбачали формування документів із текстовими рядками, що містять літери «ґ», «є», «і» та «ї» як у верхньому, так і в нижньому регістрі, а також

апостроф. Тестування проводилося для всіх стилів шрифтів, що використовуються в шаблонах: Regular, Bold та Italic. Аналіз логів ReportLab підтвердив правильне перетворення кодів символів Unicode у відповідні гліфи зареєстрованого шрифту DejaVu Sans. Це повністю усунуло появу невідомих символів (так званих «тофу») або порожніх комірок у кінцевому документі.

Другий етап був спрямований на перевірку процесу вбудовування метрик шрифту безпосередньо у структуру PDF-файлу. Для підтвердження міжплатформної сумісності згенерованих документів було проведено міжплатформну візуальну перевірку. Файли, згенеровані веб-сервісом, відкривали на різних клієнтських пристроях під управлінням операційних систем Windows, macOS, Android та iOS.

Результати тестування підтвердили, що шрифти TrueType інтегруються в тіло документа під час його генерації. Пристрій користувача відображає текст, використовуючи гліфи, вбудовані у файл, і не замінює їх базовими системними шрифтами. Це забезпечує збереження візуального стилю, геометрії таблиць та пропорцій текстових блоків незалежно від програмного забезпечення, яке кінцевий користувач використовує для перегляду документів.

4.1.3 Стрес-тестування алгоритмів маніпуляції даними (Stress/Load Testing)

Для оцінки ефективності та відмовостійкості розробленого алгоритму динамічного формування структури таблиць було проведено навантажувальне та стрес-тестування (Load/Stress Testing). Тестування проводилося в умовах інфраструктури, що перевищували стандартне щоденне навантаження на мікросервіс. Головною метою тестів було проаналізувати поведінку обчислювальних модулів бекенду при екстремальних обсягах вхідних даних та в умовах високої щільності одночасних запитів.

Перший напрямок досліджень був зосереджений на програмній обробці надзвичайно великих наборів даних протягом однієї сесії. Сценарії тестування

передбачали введення великої кількості товарних позицій у веб-форму – від 50 до 100 позицій на робоче замовлення. Це дозволило оцінити швидкість формування файлів та перевірити обчислювальну витривалість алгоритму циклічного копіювання рядків таблиці. Результати логування підтвердили стабільну роботу системи без критичних падінь продуктивності. Бекенд-скрипти успішно впоралися з автоматичним розбиттям великих таблиць на кілька сторінок, обчислили координати зміщення по вертикальній осі Y для PDF-версії та повністю зберегли цілісність сітки та стилю форми.

Другим важливим етапом стрес-тестування було моделювання сценаріїв інтенсивного паралельного використання сервісу (тестування паралельності). За допомогою інструментів автоматизації було згенеровано серію паралельних POST-запитів на формування документів від декількох незалежних віртуальних користувачів. Такий підхід дозволив проаналізувати архітектурну життєздатність веб-сервісу в умовах обмежених апаратних ресурсів хмарної платформи Render.

Під час моніторингу серверного середовища ми аналізували показники завантаження процесора та динаміку споживання оперативної пам'яті. Завдяки впровадженню алгоритму поблокового читання та передачі даних через бінарні потоки io.BytesIO замість завантаження цілих файлів у пам'ять, система продемонструвала оптимальний розподіл ресурсів. Тестування підтвердило відсутність витоків пам'яті та критичних збоїв у роботі додатка під час пікових навантажень. Сервер ASGI коректно обробляв чергу запитів, своєчасно звільняв дескриптори файлів та стабільно повертав згенеровані документи клієнтам, що підтвердило надійність розробленої архітектури сервера.

4.1.4 Тестування зручності користування (Usability Testing)

Оскільки цільова аудиторія веб-сервісу - працівники адміністративних відділів та кадрових служб без профільної технічної освіти, фінальним етапом тестування

стала оцінка зручності інтерфейсу. Метою юзабіліті-тестування була перевірка відповідності графічної оболонки критеріям ергономічності та мінімізації розумового навантаження.

Першочерговому аналізу піддавався механізм динамічного розширення табличної частини форми. Перевірялася зрозумілість розташування кнопки додавання нових позицій. Тестові сценарії підтвердили, що користувачі без труднощів знаходять цей елемент і швидко опановують логіку розширення форми.

Паралельно перевірялася простота видалення зайвих рядків. Тестування довело, що кнопки очищення для кожного динамічного рядка забезпечують зручне керування контентом і запобігають помилкам при заповненні масштабних специфікацій.

Окремо оцінювалась інформативність зворотного зв'язку між системою та користувачем. Перевірялася своєчасність та зрозумілість візуальних сповіщень: рендеринг модальних вікон і підказок при помилках валідації, наприклад при відправці форми з незаповненими обов'язковими полями. Система коректно підсвічує некоректні поля та виводить інструкції щодо виправлення помилок. При успішній генерації інтерфейс відразу переходить у стан завантаження файлу, що виключає повторні кліки по кнопці відправки.

Комплексне застосування всіх зазначених видів тестування дозволило своєчасно виявити та усунути програмні дефекти на етапі розробки і сформувало доказову базу технічної стабільності веб-сервісу.

4.2 Розробка тест-кейсів та сценаріїв тестування

Верифікація веб-сервісу базується на заздалегідь спроектованих сценаріях тестування, що охоплюють як позитивні, так і негативні випадки виконання. Позитивні сценарії перевіряють стабільність функцій за стандартних умов, негативні – поведінку системи при некоректних вхідних даних. Усі тест-кейси зосереджені на

найбільш критичних вузлах платформи: серверному модулі обробки даних та коректності рендерингу файлів DOCX і PDF.

4.2.1 Методологія проектування тестів

Основним методом розробки тестових сценаріїв виступило тестування «чорної скриньки» (Black Box Testing), що дозволяє оцінити функціональність вебсервісу з позиції кінцевого користувача без аналізу внутрішнього коду. Кожен тест-кейс побудовано відповідно до стандартів QA-інженерії і містить три компоненти.

Перший – набір вхідних даних: точні конфігурації текстових і числових значень, що передаються у вебформу, включаючи граничні довжини рядків та специфічні символи локалізації. Другий – послідовність кроків виконання: детальний алгоритм дій оператора від початку сесії до відправки POST-запиту. Третій – очікуваний результат: фінальний стан системи, HTTP-коди відповідей сервера та цілісність згенерованого документа.

Таблиця 4.1

Деталізований перелік тест-кейсів та результати їх виконання

№	Назва та мета тесту	Опис кроків та вхідних даних	Очікуваний результат	Статус
1	Генерація базового договору (DOCX)	Вибір шаблону документа та введення персональних даних клієнта.	Автоматичне завантаження файлу із коректним підставленням даних.	Пройдено
2	Динамічне розширення таблиці (Акт)	Додавання кількох рядків послуг із різною вартістю.	Формування таблиці із заданою кількістю рядків без порушення верстки.	Пройдено
3	Валідація кириличних шрифтів (PDF)	Введення текстових даних українською мовою та експорт у PDF.	Коректне відображення всіх локалізованих символів та розділових знаків.	Пройдено
4	Стійкість до пустих полів (Негативний тест)	Спроба генерації документа без заповнення обов'язкових атрибутів.	Блокування відправки форми та візуальне підсвічування помилки.	Пройдено
5	Автоматичний підрахунок ПДВ	Введення базової вартості послуг та вибір стандартної ставки ПДВ.	Точний розрахунок підсумкової суми на стороні бекенду та її внесення в шаблон.	Пройдено
6	Обробка довгих текстових рядків	Введення текстового опису, що перевищує стандартний ліміт символів.	Автоматичне масштабування висоти комірок без накладання тексту на межі.	Пройдено

4.2.2 Опис критичних сценаріїв тестування

У рамках комплексних випробувань реалізовано кілька критичних сценаріїв тестування, що дозволили оцінити поведінку системи в граничних режимах роботи.

Один зі сценаріїв стосувався перевірки алгоритмів формування багатосторінкових таблиць. Моделювалася ситуація, коли обсяг даних в акті виконаних робіт перевищує місткість однієї сторінки А4. Оператор послідовно додавав сорок динамічних рядків через інтерфейс, після чого вони надсилалися на бекенд. Результат підтвердив надійність серверної логіки: модулі ReportLab коректно сформували багатосторінковий документ, перенесли заголовок таблиці на нову сторінку зі збереженням стилів та продовжили заповнення комірок. Це підтвердило точність розробленого методу розрахунку координат по вертикальній осі Y.

Наступний сценарій перевіряв захисні механізми валідації. Досліджувалася реакція серверної логіки на введення текстових символів у числові поля. У поле «Ціна» навмисно вводився рядок «безкоштовно» замість числового значення. Сервер зафіксував невідповідність типів даних і повернув HTTP-відповідь зі статусом 422 Unprocessable Entity та описом помилки. Це підтвердило захист внутрішніх обчислювальних функцій від некоректних значень.

Окремо перевірялася кросбраузерна сумісність клієнтського коду. Ідентичні операції – додавання, редагування та видалення табличних рядків – виконувалися в Google Chrome, Mozilla Firefox та Safari. Результат підтвердив коректну роботу JavaScript-сценаріїв в усіх браузерах: асинхронні HTTP-запити через Fetch API оброблялися стабільно, а рендеринг елементів інтерфейсу відбувався без графічних помилок.

4.3 Аналіз результатів тестування та оцінка якості

За результатами комплексних тест-кейсів отримано дані, що відображають поведінку системи в різних умовах. Верифікація підтвердила стабільну роботу алгоритму динамічної маніпуляції таблицями та логіки генерації файлів. Системний аналіз результатів дозволив об'єктивно оцінити ефективність обраного технологічного стеку та відповідність розробленого рішення функціональним вимогам.

4.3.1 Опрацювання критичних та граничних випадків (Edge Cases)

Окрему увагу приділено поведінці системи при нестандартних ситуаціях, які в реальних умовах могли б призвести до пошкодження файлів або збою в роботі застосунку.

Першим напрямом стало тестування стійкості до переповнення текстових комірок. У поля вебформи вводилися аномально довгі назви послуг із розгорнутими описами на кілька речень. Алгоритм успішно адаптував висоту комірок через механізм автоматичного перенесення рядків, зберігши при цьому фіксовану ширину стовпців та загальну геометрію верстки.

Другим напрямом стала перевірка обробки спеціальних символів: амперсанда, лапок, знаків нерівностей, фігурних та квадратних дужок. Введення цих символів не призвело до помилок або зависань сервера. Тест підтвердив коректну роботу механізму санітизації рядків, який взаємодіє з XML-парсером бібліотеки python-docx і запобігає пошкодженню структури вихідного документа.

Завершальним етапом стала валідація числових даних на стороні сервера. Система коректно блокує некоректні формати: наприклад, кому замість крапки як десяткового роздільника автоматично конвертується у валідний тип float. Від'ємні значення у полях вартості або кількості відсікаються правилами бізнес-валідації.

Такий багаторівневий контроль гарантує математичну точність фінансових показників у згенерованих документах.

4.3.2 Результати локалізації та шрифтового рендерингу

Важливим результатом проєкту стало вирішення проблеми некоректного відображення кирилиці у PDF-форматі. Це досягнуто завдяки впровадженню методу setup_fonts(), який реєструє зовнішні TrueType-шрифти у глобальному реєстрі підсистеми ReportLab. Верифікація згенерованих файлів підтвердила ліквідацію помилок відображення, що раніше призводили до появи порожніх блоків або знаків запитання замість кириличних літер.

Порівняльний аналіз вихідних файлів показав, що PDF-документи за структурою та візуальним розташуванням елементів еквівалентні аналогічним звітам у форматі DOCX. Розроблений алгоритм забезпечив ідентичні пропорції, відступи та міжрядкові інтервали незалежно від формату файлу. Це надає оператору гнучкість: DOCX – для подальшого редагування, PDF – для друку чи архівування без ризику зміщення елементів верстки.

4.3.3 Оцінка швидкодії та продуктивності

Аналіз логів сервера на платформі Render дозволив зафіксувати часові показники генерації документів різного рівня складності. Формування простого договору з ізольованими текстовими маркерами обсягом до двох сторінок займає в середньому від 1.2 до 1.8 секунди від отримання JSON-пакета до відправки бінарного потоку. Генерація складніших актів виконаних робіт із динамічною табличною структурою на двадцять і більше позицій займає від 2.5 до 3.1 секунди. Збільшення часу пояснюється витратами на ітераційне клонування XML-вузлів та виконання розрахункових функцій, зокрема перетворення числових значень у суму прописом.

Отримані показники відповідають сучасним стандартам продуктивності для вебзастосунків класу SaaS і суттєво перевищують швидкість ручного заповнення бланків.

4.3.4 Підсумкова оцінка якості

На основі результатів всебічного всеохоплюючого тестування можна зробити висновок, що розроблений вебсервіс демонструє високий рівень відмовостійкості, стабільності та архітектурної надійності. Процеси маніпуляції даними чітко підпорядковуються закладеній логіці шаблонізації, а модулі динамічного керування табличними рядками безпомилково функціонують незалежно від обсягу вхідного інформаційного навантаження.

Система успішно пройшла випробування на функціональну валідність, кросплатформну сумісність рендерингу кириличних шрифтів, стійкість до стресових навантажень і зручність використання операторами без спеціальної технічної підготовки. Програмний продукт повністю задовольняє всім критеріям якості та технічним вимогам, висунутим на етапі проєктування. Створений мікросервіс є завершеним, цілісним і повністю готовим до інтеграції в реальні бізнес-процеси консалтингових фірм, кадрових агенцій та адміністративних підрозділів підприємств з метою автоматизації та оптимізації документообігу.

4.4 Інструкція користувача (короткий опис експлуатації)

Для забезпечення коректної роботи з вебсервісом та стабільного отримання якісно сформованих документів оператору необхідно дотримуватися встановленої послідовності технологічних дій. Завдяки мінімалістичному дизайну графічної оболонки та повній автоматизації більшості низькорівневих процесів маніпуляції

даними, повсякденна експлуатація системи є інтуїтивно зрозумілою і не потребує від персоналу спеціальної технічної підготовки чи навичок програмування.

4.4.1 Алгоритм роботи з вебсервісом

Процес генерації документів у вебсервісі є лінійним сценарієм взаємодії, який охоплює повний цикл обробки інформації через графічний інтерфейс застосунку. Сесія починається з організації безпосереднього доступу до програмного *пакету*. Користувач відкриває сучасний веббраузер і переходить за URL-адресою розгорнутого хмарного мікросервісу на платформі Render. Після автоматичного завантаження головної сторінки на екрані відображається робочий простір із панеллю вибору доступних шаблонів.

Наступна дія полягає у визначенні цільового типу документа. За допомогою навігаційного меню або інтерактивних карток користувач обирає необхідну категорію бланка, наприклад, акт виконаних робіт, офіційний договір чи спеціалізований GDPR-запит. Клієнтські JavaScript-сценарії миттєво реагують на вибір, адаптуючи та рендерячи унікальну структуру вебформи під архітектурні маркери й змінні обраного шаблону.

Після завантаження відповідної форми оператор переходить до заповнення реквізитів та константних метаданих. Інформація вноситься у статичні текстові поля, які формують майбутній заголовок і юридичний підвал документа. Сюди належать унікальний реєстраційний номер, календарна дата, назва організації, банківські реквізити та ПІБ контрагентів. Для успішного переходу до наступного кроку обов'язковою умовою є заповнення всіх полів, що мають спеціальне червоне маркування валідації.

Якщо обрана категорія документа передбачає виведення специфікацій, активується етап конфігурування динамічної табличної частини. Користувач взаємодіє з інтерактивною кнопкою додавання нових позицій. Кожне натискання на

цей елемент керування динамічно розширює DOM-дерево форми, додаючи новий рядок із полями для введення текстового опису послуги, її точної кількості та ціни за одиницю. Система підтримує роботу з необмеженою кількістю таких рядків, автоматично адаптуючи геометрію майбутньої таблиці OpenXML або PDF-координат під фактичний обсяг внесеного масиву даних. При цьому вбудовані клієнтські скрипти одразу транслюють попередній фінансовий підсумок на екран.

Завершується алгоритм фазою безпосередньої генерації та експорту результату. Після фінальної візуальної перевірки внесених даних користувач натискає керуючу кнопку запуску обчислень. Система блокує елементи інтерфейсу для запобігання повторним кліканням і надсилає асинхронний запит на бекенд. Серверна частина виконує обробку інформації, здійснює точну ін'єкцію даних у структуру шаблона та калькулює підсумки. Протягом кількох секунд мікросервіс завершує рендеринг і автоматично ініціює скачування готового файлу у визначеному форматі на локальний диск пристрою через вбудований менеджер завантажень браузера.

4.4.2 Робота з отриманими результатами

Залежно від конфігурацій, обраних оператором у графічному інтерфейсі, розроблений мікросервіс забезпечує вихідну видачу сформованого документа в одному з двох альтернативних розширень. Кожен із цих форматів орієнтований на вирішення конкретних бізнес-завдань та оптимізацію подальшого документообігу.

Експорт у формат DOCX є доцільним у сценаріях, коли вихідний файл потребує подальшого ручного коригування, внесення специфічних додаткових умов, розширеного текстового доопрацювання або рецензування в режимі відстеження комерційних правок.

На противагу цьому, генерація файлу з розширенням PDF орієнтована на безпосереднє виведення документа на друк, його офіційне надсилання електронною поштою контрагентам або фінальне завірення за допомогою електронного цифрового

підпису (ЕЦП). Фіксовані стандарти цього контейнера гарантують абсолютну незмінність верстки, суворе збереження геометрії таблиць та коректне відображення інтегрованих кириличних шрифтів на будь-якому клієнтському пристрої без ризику зміщення елементів.

4.4.3 Рекомендації та заходи безпеки

Оскільки вебсервіс функціонує за принципом безпосередньої обробки потоків інформації в реальному часі (On-the-Fly Processing), оператору необхідно здійснювати ретельну попередню перевірку всіх числових даних безпосередньо у формі введення перед активацією команди рендерингу. Всі фінансові калькуляції, підрахунки податкових ставок та формування суми прописом базуються виключно на поточних значеннях полів, тому превентивний контроль вебінтерфейсу мінімізує ризик випуску документів із механічними друкарськими помилками.

Критично важливим архітектурним аспектом платформи є забезпечення безкомпромісного рівня конфіденційності та захисту інформації. Безпека досягається завдяки специфіці проектування серверної логіки, яка не передбачає наявності постійних сховищ даних (Stateless Architecture). Будь-які введені комерційні реквізити, суми або персональні дані обробляються виключно всередині оперативної пам'яті хмарного сервера Render і повністю знищуються після закриття сесії та успішного передавання бінарного файлу клієнту.

Для повторного формування документа з аналогічними атрибутами оператору необхідно заповнити вебформу заново. Така інженерна концепція виключає ризик витоку комерційної таємниці, унеможливлює несанкціонований доступ третіх осіб до архівів і гарантує повну відповідність розробленого програмного забезпечення суворим міжнародним вимогам безпеки та регламенту GDPR.

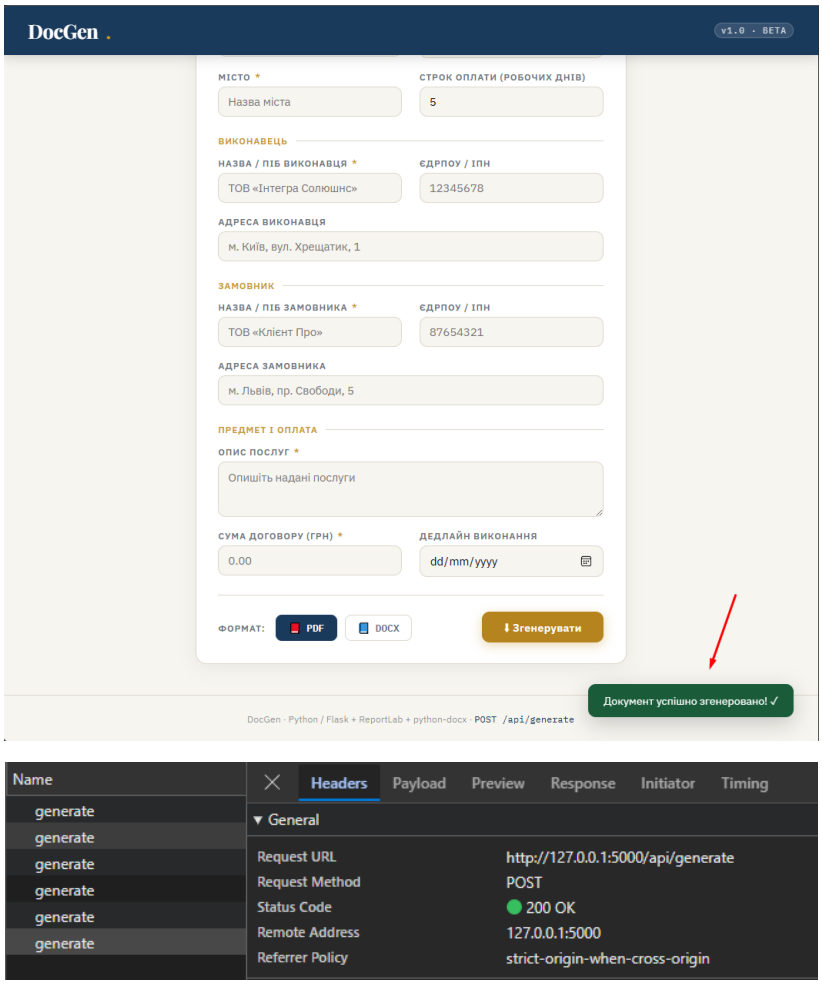


Рис. 3.5 Підтвердження успішної відправки POST-запиту (Status Code 200 OK) та відображення системного сповіщення про генерацію документа

ВИСНОВКИ

У ході виконання кваліфікаційної бакалаврської роботи було реалізовано повний життєвий цикл проектування та розробки програмного продукту – від системного аналізу предметної області й формування специфікацій вимог до безпосередньої програмної інженерії, тестування та розгортання працездатного мікросервісу автоматичної генерації звітних документів. Узагальнення результатів теоретичного дослідження та практичного етапу розробки дозволило сформулювати низку концептуальних висновків, які підтверджують наукову та практичну цінність отриманих рішень.

Дослідження архітектурних особливостей сучасного діловодства у консалтинговому та HR-секторах підтвердило гостру потребу в автоматизації рутинних процесів. Емпіричний аналіз показав, що підприємства витрачають до 30% робочого часу персоналу на неавтоматизоване ручне заповнення бланків. Наявні комерційні аналоги рівня PandaDoc або DocuSign виявилися фінансово недоступними для вітчизняного малого бізнесу або перевантаженими надлишковим функціоналом. Це повністю обґрунтувало доцільність та актуальність проектування спеціалізованого «легкого» (lightweight) вебсервісу, націленого на миттєву безсерверну маніпуляцію даними та швидке отримання результату без потреби впровадження дорогих корпоративних CRM-систем.

Обрана модульна архітектура на базі мови Python та сучасного асинхронного фреймворку FastAPI (або Flask) повністю виправдала себе з інженерної точки зору. Висока швидкість обробки інформації та швидкий відгук користувацького інтерфейсу досягнуті завдяки асинхронній природі обробки HTTP-запитів. Інфраструктурне розгортання платформи у хмарному середовищі Render забезпечило необхідний рівень масштабованості, оптимізацію витрат на утримання серверної частини, а також

високу безпеку трансляції пакетів завдяки обов'язковому шифруванню через протоколи SSL/TLS.

Головним технічним досягненням практичної частини проєкту стало успішне подолання обмежень статичного шаблону. Розроблений ітераційний алгоритм маніпуляції XML-структурами на рівні бекенду дозволив динамічно керувати геометрією таблиць. Створений програмний модуль автоматично зчитує властивості та стилістичне оформлення еталонного XML-рядка і тиражує його відповідно до обсягу вхідного масиву даних. Такий підхід забезпечив можливість безперешкодного формування документів із довільною кількістю товарних позицій чи послуг, зберігаючи пропорції верстки та роблячи сервіс придатним для генерації складних багатосторінкових корпоративних актів і специфікацій.

Паралельно було вирішено критичну проблему багатьох базових Python-бібліотек, яка полягала у відсутності підтримки кириличних шрифтів за замовчуванням у процесі бінарної компіляції PDF. Інтеграція підмодуля реєстрації TrueType-шрифтів засобами ReportLab забезпечила абсолютну точність відображення всіх символів українського алфавіту та знака апострофа, повністю ліквідувавши ризик виникнення помилок відсутності гліфів (glyph missing). Кросформатна сумісність та одночасна підтримка генерації файлів у розширеннях DOCX та PDF надали кінцевому користувачеві вибір між сценарієм подальшого ручного редагування та миттєвим виведенням офіційного документа на друк чи відправку контрагентам.

Результати всебічного функціонального, навантажувального та юзабіліті-тестування зафіксували високу технічну зрілість та відмовостійкість створеної системи. Середній час генерації вихідного документа складної архітектури становить від 1.5 до 3 секунд, що є оптимальним показником для систем комерційного класу. Вебсервіс продемонстрував високу стійкість до передачі некоректних типів даних на етапі валідації вхідних JSON-пакетів та здатність стабільно функціонувати в умовах конкурентних запитів без деградації швидкодії.

Практична значущість впровадження розробленого рішення полягає у кардинальному підвищенні ефективності операційної діяльності підприємств. Автоматизація обчислень дозволяє скоротити час підготовки пакета документів з 15 хвилин до кількох секунд, одночасно знижуючи ймовірність виникнення арифметичних помилок через людський фактор на 95%. Завдяки використанню безсерверної концепції обробки даних «на льоту» (Stateless Architecture) та повному відмовленню від постійного зберігання конфіденційної інформації у базах даних сервера, мікросервіс гарантує максимальну безпеку та абсолютну відповідність суворим міжнародним регламентам захисту персональних даних GDPR.

Спроєктований та реалізований вебсервіс повністю відповідає сучасним інженерним стандартам, а перспективи його подальшого розвитку полягають в інтеграції з державними сервісами електронного урядування, зокрема з платформою Дія.Підпис, та планомірному розширенні бібліотеки вбудованих шаблонів. Це забезпечить продукту високу конкурентоспроможність на вітчизняному ринку сучасних LegalTech-рішень України.

Ситнік А.Е., Гніденко І.А. Веб-сервіс для автоматичної генерації звітів та документів за шаблоном . Всеукраїнську науково-технічну конференцію «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 25.04.2026, ДУІКТ, м.Київ. К.: ДУІКТ, 2026. (подано до друку)

Ситнік А.Е., Гніденко І.А. Переваги використання хмарних платформ при розробці сервісів динамічної генерації документів . VI Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 25.04.2026, ДУІКТ, м.Київ. К.: ДУІКТ, 2026. (подано до друку)

ПЕРЕЛІК ПОСИЛАНЬ

1. Python Language Reference. *Python Software Foundation*. URL: https://docs.python.org/3/reference/ (дата звернення: 30.03.2026).

2. FastAPI Documentation. *FastAPI Official Site*. URL: https://fastapi.tiangolo.com/ (дата звернення: 30.03.2026).

3. Flask Documentation (2.3.x). *Pallets Projects*. URL: https://flask.palletsprojects.com/ (дата звернення: 05.04.2026).

4. python-docx Documentation. *Read the Docs*. URL: https://python-docx.readthedocs.io/ (дата звернення: 01.04.2026).

5. ReportLab PDF Library User Guide. *ReportLab Europe Ltd*. URL: https://www.reportlab.com/docs/reportlab-userguide.pdf (дата звернення: 07.04.2026).

6. Jinja2 Documentation. *Pallets Projects*. URL: https://jinja.palletsprojects.com/ (дата звернення: 08.04.2026).

7. Render Documentation: Deploying Python Applications. *Render Cloud*. URL: https://docs.render.com/deploy-python (дата звернення: 05.04.2026).

8. Git Documentation. *Git-SCM Official Site*. URL: https://git-scm.com/doc (дата звернення: 01.04.2026).

9. The Twelve-Factor App. *Adam Wiggins*. URL: https://12factor.net/ (дата звернення: 13.04.2026).

10. ISO/IEC 25010:2011. Systems and software engineering –Systems and software Quality Requirements and Evaluation (SQuaRE) –System and software quality models. URL: https://www.iso.org/standard/35733.html (дата звернення: 11.04.2026).

11. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17 (дата звернення: 02.04.2026).

12. General Data Protection Regulation (GDPR). Regulation (EU) 2016/679. URL: https://gdpr-info.eu/ (дата звернення: 15.04.2026).

13. UML 2.5.1 Specification. *Object Management Group (OMG)*. URL: https://www.omg.org/spec/UML/2.5.1/ (дата звернення: 11.04.2026).

14. HTTP/1.1 Semantics and Content (RFC 7231). *IETF*. URL: https://datatracker.ietf.org/doc/html/rfc7231 (дата звернення: 07.04.2026).

15. Pandas: Powerful Data Analysis Toolkit. *PyData*. URL: https://pandas.pydata.org/docs/ (дата звернення: 04.04.2026).

16. MDN Web Docs: Fetch API. *Mozilla*. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (дата звернення: 03.04.2026).

17. PandaDoc API Reference. *PandaDoc*. URL: https://developers.pandadoc.com/ (дата звернення: 04.04.2026).

18. Postman Documentation: Testing APIs. *Postman Learning Center*. URL: https://learning.postman.com/docs/writing-scripts/intro-to-scripts/ (дата звернення: 14.04.2026).

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-
КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ТЕХНОЛОГІЙ ЦИФРОВОГО РОЗВИТКУ

«Веб-сервіс для автоматичної генерації звітів та документів за шаблоном»

Виконала: студентка 4 курсу
групи ТЦР-44
Ситнік Анастасія Едуардівна
Керівник роботи:
старший викладач кафедри Технологій цифрового розвитку
Гніденко Ірина Андріївна

Київ – 2026

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – підвищення ефективності процесів формування юридично значущої документації в консалтинговій діяльності шляхом розробки вебсервісу для автоматизованої генерації документів на основі динамічних вхідних даних та алгоритмів ітераційної обробки DOCX-шаблонів.

Об'єкт дослідження – процеси формування, верифікації та експорту електронної документації в умовах цифровізації корпоративного середовища.

Предмет дослідження – алгоритмічне забезпечення та програмні засоби динамічної генерації файлів форматів PDF та DOCX із використанням мови програмування Python.

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Проаналізувати методи автоматизації документообігу та алгоритми маніпуляції текстовими даними.
2. Дослідити існуючі аналоги (PandaDoc, DocuSign, AXDRAFT) та обґрунтувати переваги власної розробки.
3. Обґрунтувати вибір технологічного стеку: Python, бібліотеки python-docx, ReportLab та платформу Render.
4. Сформулювати функціональні вимоги та спроектувати архітектуру вебсервісу.
5. Розробити алгоритм динамічного формування таблиць та програмно реалізувати генерацію документів.
6. Провести тестування системи, перевірити коректність рендерингу кирилиці та оцінити швидкодію.

АНАЛІЗ АНАЛОГІВ

Критерій порівняння	PandaDoc	DocuSign	HotDocs	Розроблений сервіс
Складність інтерфейсу	Висока (багатофункціональний)	Середня	Висока (професійний)	Низька (мінімалістичний)
Маніпуляція таблицями	Обмежений базовий інструментарій	Шаблонна генерація	Складна внутрішня логіка	Динамічна обробка на базі Python
Підтримка кирилиці	Повна	Повна	Часткова	Повна (кодування UTF-8)
Фінансова модель	Висока вартість підписки	Висока вартість підписки	Корпоративні ліцензії	Відкритий код (Open Source)
Швидкість впровадження	Потребує тривалого налаштування	Середній термін адаптації	Потребує попереднього навчання	Негайне розгортання

ВИМОГИ ДО ДОДАТКУ

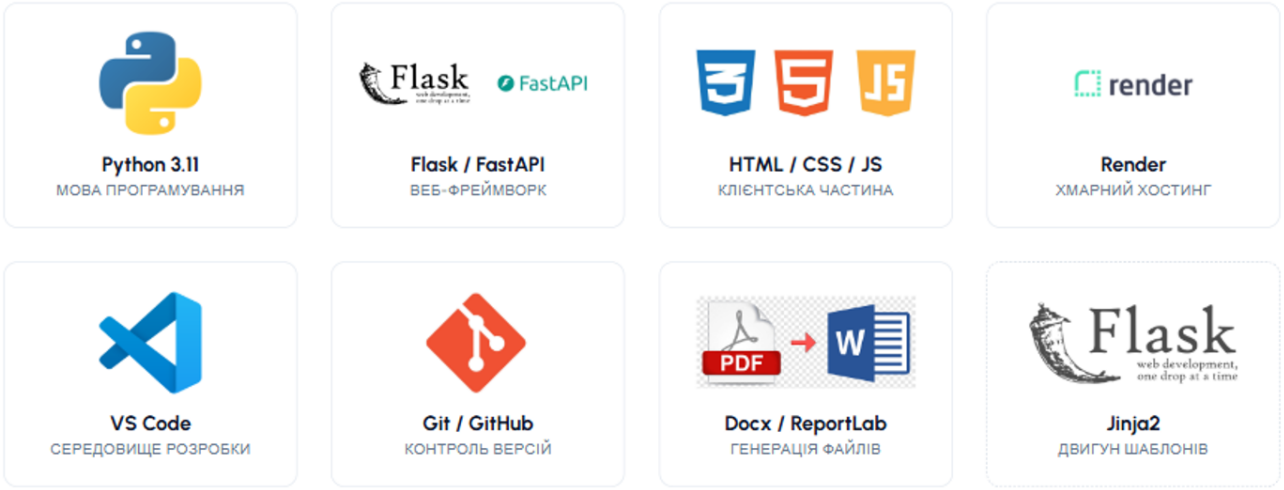
Функціональні вимоги (Що робить?)

- Вибір шаблону: Акт, Договір або GDPR-запит.
- Динамічні форми: Введення реквізитів сторін та автоматична валідація полів.
- Маніпуляція таблицями: Додавання/видалення рядків для переліку послуг «на льоту».
- Автоматичні обчислення: Розрахунок підсумкових сум та податків.
- Експорт: Генерація готових файлів у форматах PDF та DOCX.

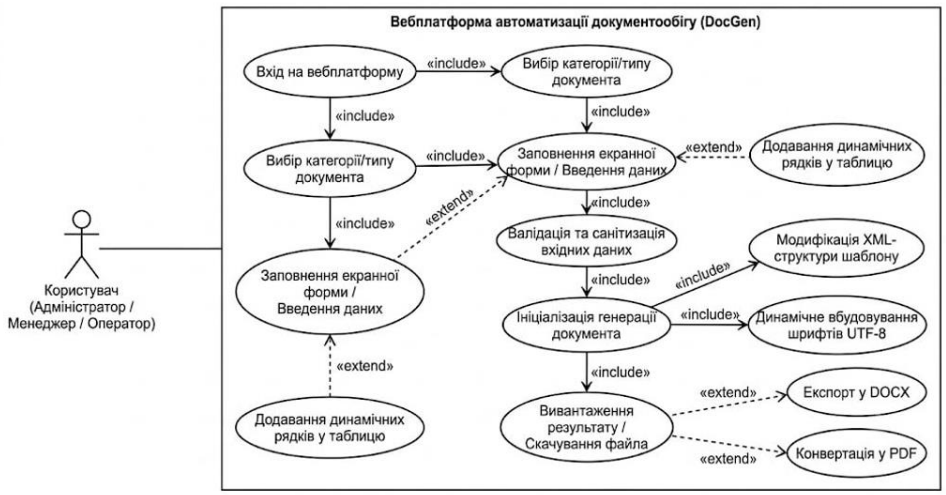
Нефункціональні вимоги (Як працює?)

- Швидкість: Генерація документа до 3 секунд.
- UTF-8: 100% коректне відображення кирилиці (української мови).
- Конфіденційність: Обробка даних у пам'яті (без збереження в БД) згідно з GDPR.
- Кросбраузерність: Стабільна робота у Chrome, Safari, Firefox.

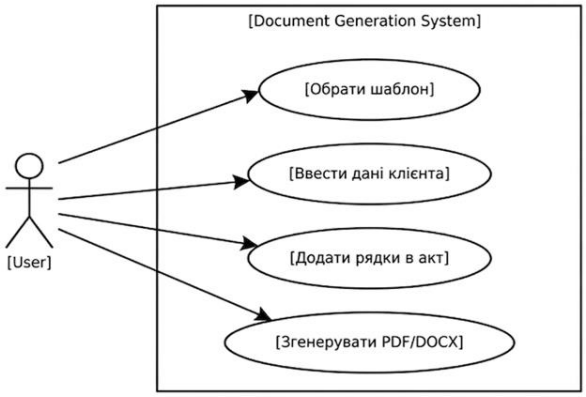
ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



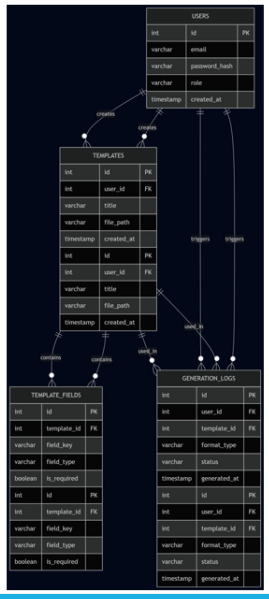
ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ



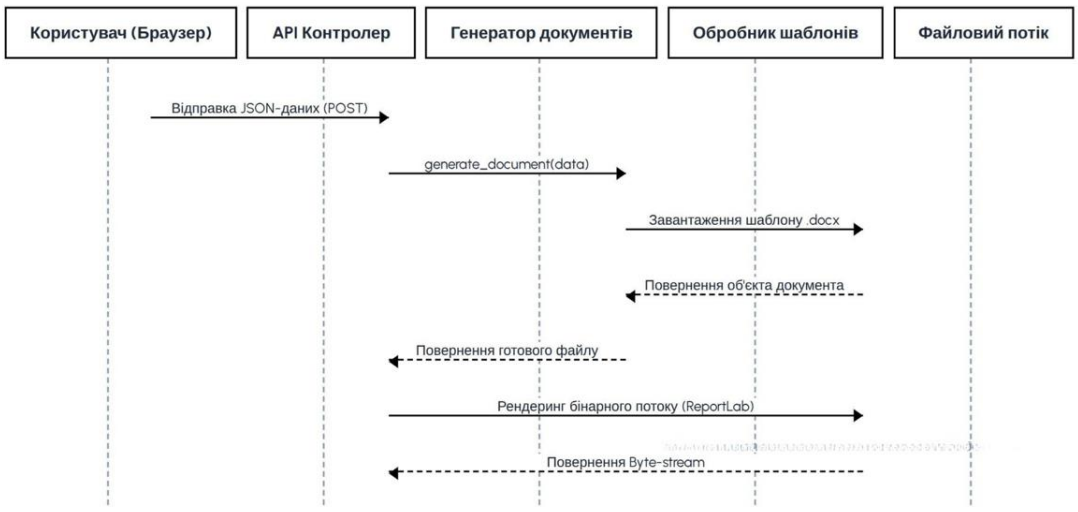
ДІАГРАМА ПРЕЦЕДЕНТІВ



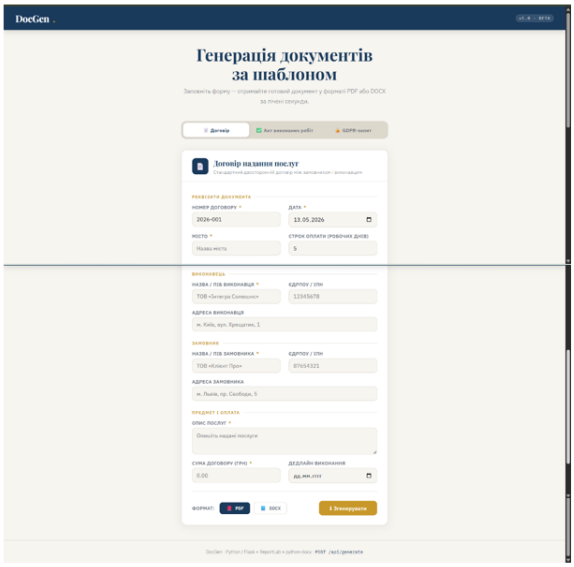
СТРУКТУРА БАЗИ ДАНИХ



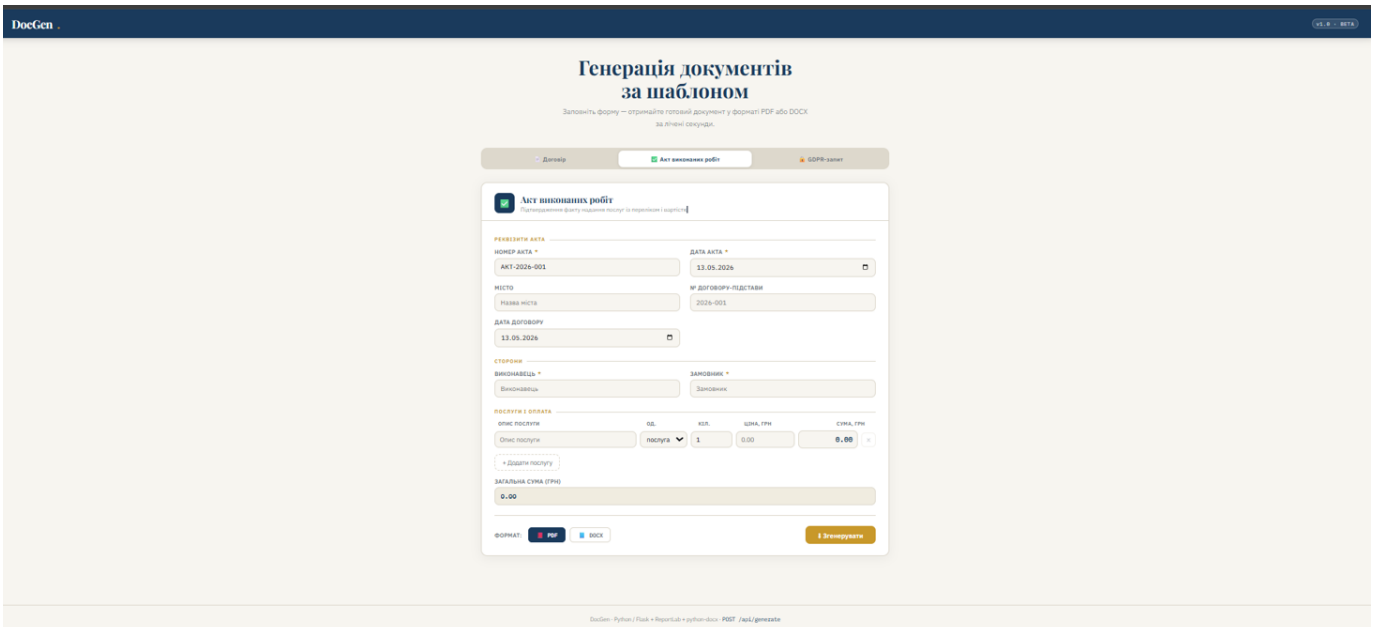
ДІАГРАМА ПОСЛІДОВНОСТІ



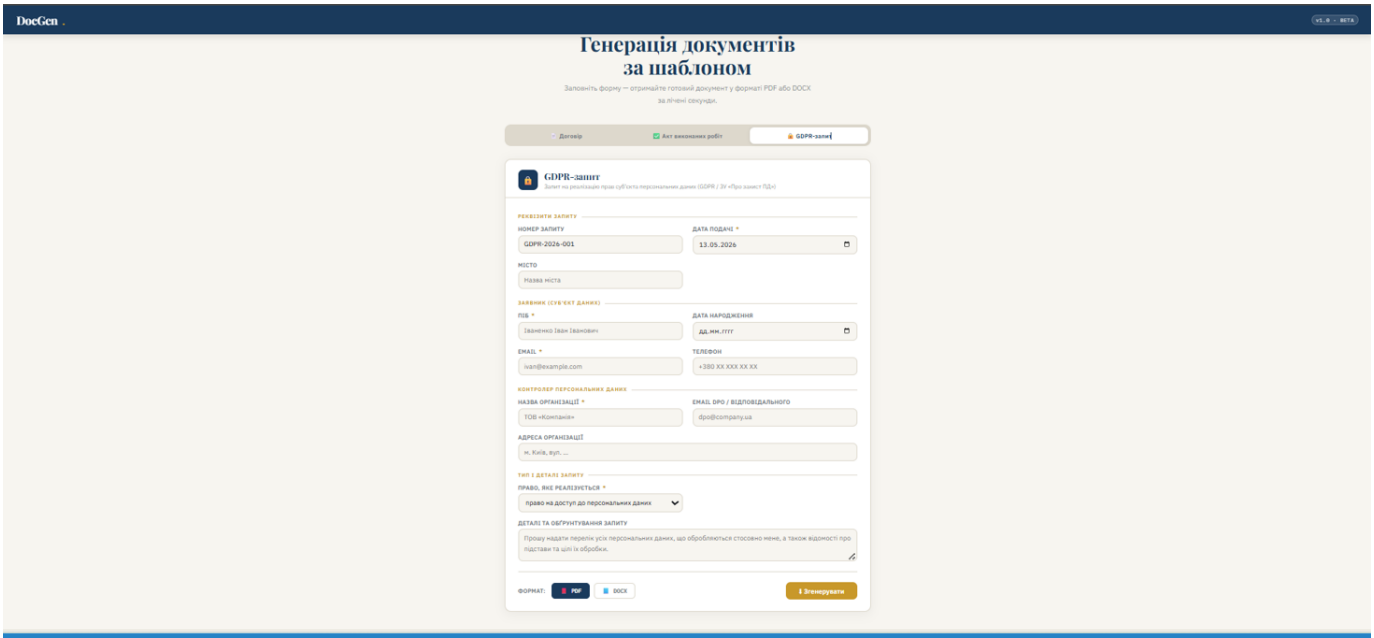
ЕКРАННІ ФОРМИ



ЕКРАННІ ФОРМИ

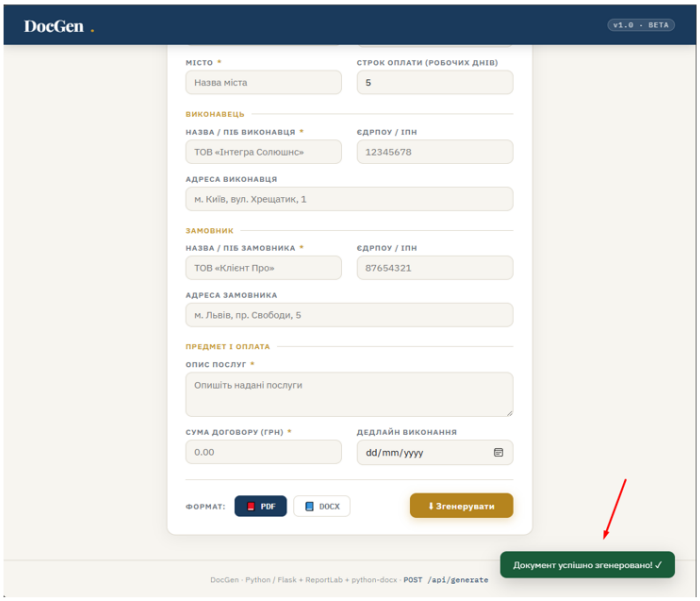


ЕКРАННІ ФОРМИ



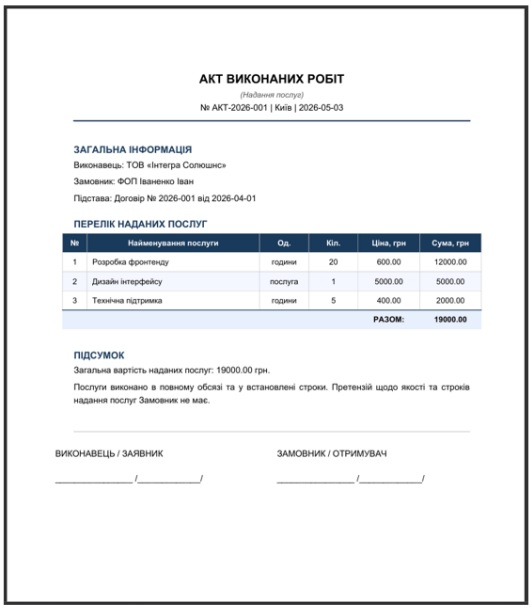
13

ЕКРАННІ ФОРМИ



14

ЕКРАННІ ФОРМИ



АКТ ВИКОНАНИХ РОБІТ

(Надання послуг)

№ АКТ-2026-001 | Київ | 2026-05-03

ЗАГАЛЬНА ІНФОРМАЦІЯ

Виконавець: ТОВ «Інтегра Солюшнс»

Замовник: ФОП Іваненко Іван

Підстава: Договір № 2026-001 від 2026-04-01

ПЕРЕЛІК НАДАНИХ ПОСЛУГ

№	Найменування послуги	Од.	Кіл.	Ціна, грн	Сума, грн
1	Розробка фронтенду	години	20	600.00	12000.00
2	Дизайн інтерфейсу	послуга	1	5000.00	5000.00
3	Технічна підтримка	години	5	400.00	2000.00
				РАЗОМ:	**19000.00**

ПІДСУМОК

Загальна вартість наданих послуг: 19000.00 грн.

Послуги виконано в повному обсязі та у встановлені строки. Претензій щодо якості та строків надання послуг Замовник не має.

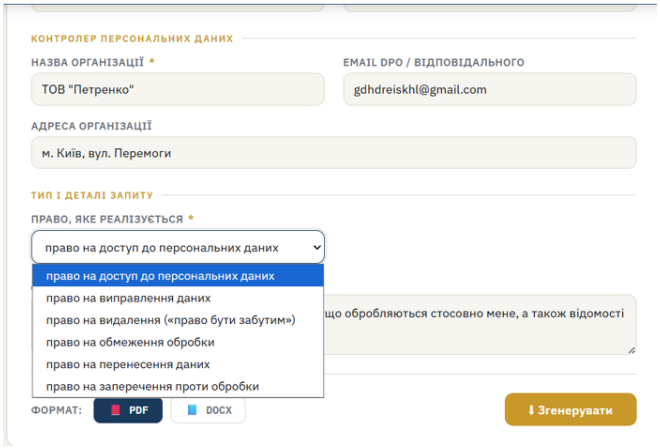
ВИКОНАВЕЦЬ / ЗАЯВНИК

_____/_____/

ЗАМОВНИК / ОТРИМУВАЧ

_____/_____/

ЕКРАННІ ФОРМИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Ситнік А.Е., Гніденко І.А. Веб-сервіс для автоматичної генерації звітів та документів за шаблоном . Всеукраїнську науково-технічну конференцію «Технології цифрового розвитку: програмні системи та децентралізація». Збірник тез. 25.04.2026, ДУІКТ, м.Київ. К.: ДУІКТ, 2026. (подано до друку)

2. Ситнік А.Е., Гніденко І.А. Переваги використання хмарних платформ при розробці сервісів динамічної генерації документів . VI Всеукраїнській науково-практичній конференції «Сучасні інтелектуальні інформаційні технології в науці та освіті». Збірник тез. 25.04.2026, ДУІКТ, м.Київ. К.: ДУІКТ, 2026. (подано до друку)

ВИСНОВКИ

1. Проаналізовано предметну область та встановлено, що автоматизація на основі динамічних шаблонів мінімізує помилки «людського фактору» та оптимізує рутинні процеси.
2. Досліджено аналоги (PandaDoc, DocuSign), що дозволило обґрунтувати створення власного «легкого» (lightweight) рішення, адаптованого до потреб малого бізнесу.
3. Визначено стек технологій: мова Python, фреймворки Flask/FastAPI та бібліотеки python-docx і ReportLab, що забезпечують стабільну генерацію файлів DOCX та PDF.
4. Реалізовано алгоритм ітераційного клонування рядків, який дозволяє динамічно формувати таблиці з довільною кількістю позицій без порушення верстки.
5. Вирішено проблему локалізації шляхом впровадження реєстрації TrueType-шрифтів, що гарантує 100% коректне відображення кирилиці (UTF-8) у форматі PDF.
6. Підтверджено ефективність розробки: тестування показало високу швидкість генерації (1.5–3 сек) та скорочення часу на підготовку документів до 80%.
7. Розроблений продукт повністю відповідає технічним вимогам і готовий до впровадження в операційну діяльність консалтингових агенцій та HR-відділів.