МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ОНЛАЙН СЕРВІСУ КОНВЕРТАЦІЇ ДОКУМЕНТІВ З ВИКОРИСТАННЯМ ХМАРНИХ ОБЧИСЛЕНЬ»**

Виконав: студент 4 курсу, групи 6.1211-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
Д.О. Філь
(ініціали та прізвище)

Керівник завідувач кафедри програмної інженерії, доцент, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Борю С.Ю.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Філю Даніілу Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка онлайн сервісу конвертації документів з використанням хмарних обчислень

керівник роботи Лісняк Андрій Олександрович, к.ф-м.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 02.06.2025 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд технологій та технічне завдання.
2. Проєктування системи.
3. Реалізація та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 26.12.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	08.01.2025	
2.	Збір вихідних даних.	07.02.2025	
3.	Обробка методичних та теоретичних	20.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	07.04.2025	
5.	Розробка третього розділу.	21.05.2025	
6.	Оформлення та нормоконтроль	26.05.2025	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	19.06.2025	

Студент _____ Д.О. Філь
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.О. Лісняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка онлайн сервісу конвертації документів з використанням хмарних обчислень»: 46 с., 17 рис., 3 табл., 10 джерел.

ВЕБДОДАТОК, КОНВЕРТАЦІЯ ДОКУМЕНТІВ, МАСШТАБОВАНІСТЬ, ОНЛАЙН СЕРВІС, ОПТИМІЗАЦІЯ РЕСУРСІВ, ПРОДУКТИВНІСТЬ СИСТЕМИ, ФОРМАТУВАННЯ ФАЙЛІВ, ХМАРНА ІНФРАСТРУКТУРА, ХМАРНІ ОБЧИСЛЕННЯ, API.

Об’єкт дослідження – процес розробки онлайн сервісу для конвертації документів з використанням хмарних обчислень.

Мета роботи – розробити ефективний, зручний та масштабований онлайн сервіс для конвертації документів, що використовує хмарні технології для обробки даних.

Метод дослідження – аналітичний метод для вивчення існуючих рішень, порівняльний метод для оцінки технологій, емпіричний метод для тестування створеного програмного продукту, а також експериментальний метод для перевірки його продуктивності в умовах реального використання.

У дипломній роботі проаналізовано підходи до розробки онлайн-сервісу конвертації документів з використанням хмарних технологій. Розглянуто функціональні вимоги, обґрунтовано вибір технологій, реалізовано прототип, проведено тестування ефективності, масштабованості та безпеки.

SUMMARY

Bachelor’s qualifying paper “Development of an Online Service for Document Converting Using Cloud Computing”: 46 pages, 17 figures, 3 tables, 10 references.

API, CLOUD COMPUTING, CLOUD INFRASTRUCTURE, DOCUMENT CONVERSION, FILE FORMATTING, ONLINE SERVICE, RESOURCE OPTIMIZATION, SCALABILITY, SYSTEM PERFORMANCE, WEB APPLICATION.

The object of the study is the development process of an online document conversion service utilizing cloud computing technologies.

The aim of the study is to design an efficient, scalable and user-friendly online service for document conversion, based on cloud-based solutions.

The research methods include analytical method for studying existing solutions, comparative methods for evaluating technologies, empirical methods for testing the developed software, and experimental methods for assessing system performance under real-world conditions.

This thesis presents an analysis of approaches to the development of an online document conversion service using cloud technologies. The functional requirements of the system are defined, the choice of implementation technologies is substantiated, a prototype of the software is developed, and testing is conducted to evaluate its efficiency, scalability, and security.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Огляд технологій та технічне завдання 9
1.1 Функціональні вимоги..... 9
1.1.1 Призначення і цілі створення системи 9
1.1.2 Загальні функціональні можливості системи 9
1.2 Нефункціональні вимоги..... 10
1.2.1 Інтерфейс користувача 10
1.2.2 Підтримка браузерів 10
1.2.3 Вимоги до продуктивності..... 11
1.2.4 Вимоги до безпеки 11
1.3 Опис предметної області 11
1.4 Опис системи..... 12
1.5 Порівняння з подібними системами..... 13
1.5.1 Аналіз провідних хмарних платформ 13
1.5.2 Порівняння з аналогами для конвертації документів 14
1.5.3 Унікальність та переваги розробленої системи..... 14
2 Проєктування системи..... 16
2.1 Використання UML під час розробки системи..... 16
2.2 Діаграма варіантів використання 17
2.2.1 Опис варіантів використання..... 18
2.3 Діаграма послідовності..... 21
2.4 Діаграма класів..... 22
2.5 Діаграма компонентів..... 24
2.6 Програмний інтерфейс API..... 26

3 Реалізація та тестування 29
3.1 Реалізація програмного забезпечення 29
3.1.1 Реалізація API Gateway 30
3.1.2 Реалізація компонентів асинхронної обробки документів 32
3.1.3 Реалізація вебінтерфейсу 36
3.1.4 Реалізація механізму сповіщень 37
3.2 Тестування системи 38
3.2.1 Аналіз швидкодії та продуктивності 41
3.3 Контейнеризація додатків 42
Висновки 45
Перелік посилань 46

ВСТУП

Стрімкий розвиток інформаційних технологій змінює підходи до обробки та передачі даних у сучасному цифровому суспільстві. Хмарні обчислення забезпечують доступ до обчислювальних ресурсів через мережу Інтернет, знижуючи витрати на інфраструктуру та надаючи необхідну масштабованість у роботі з інформацією.

Поширення електронного документообігу та віддаленої роботи зумовлює потребу у ефективних інструментах конвертації документів між форматами. Традиційні настільки застосунки поступаючись веборієнтованим рішенням, які забезпечують універсальний доступ незалежно від платформи чи місця розташування користувача. Таким чином, актуальність теми обумовлена цифровою трансформацією бізнес-процесів та зростання вимог до оперативності обробки інформації. Результати дослідження можуть застосовуватися для модернізації документообігу в корпоративному та державному секторах.

Метою дослідження є створення масштабованого онлайн-сервісу для конвертації документів на базі хмарних технологій. Для досягнення поставленої мети необхідно вирішити наступні задач:

– аналіз існуючих рішень;
– проєктування архітектури системи;
– вибір технологій;
– розробку користувацького інтерфейсу та серверної частини;
– тестування застосунку.

Об’єктом дослідження є процес розробки онлайн-сервісу для конвертації документів. Предметом дослідження є архітектурні рішення вебзастосунків, технології хмарної обробки даних та методи забезпечення продуктивності й безпеки сервісу.

1 ОГЛЯД ТЕХНОЛОГІЙ ТА ТЕХНІЧНЕ ЗАВДАННЯ

1.1 Функціональні вимоги

1.1.1 Призначення і цілі створення системи

Функціональне призначення системи – реалізувати можливість онлайн-конвертації документів із використанням хмарних технологій. Реалізації даної функції дозволяє підвищити зручність доступу до сервісу, скоротити витрати часу на виконання рутинних операцій та оптимізувати процес взаємодії з документами в умовах цифрової трансформації.

Експлуатаційне призначення системи – використання системи передбачено кінцевими користувачами в рамках організації або індивідуально, у випадках, коли виникає потреба у швидкому та доступному сервісі для обробки документів. Робота системи не залежить від типу пристроїв чи операційної системи, оскільки взаємодія здійснюється через вебінтерфейс.

Мета створення системи – розробка ефективного, надійного та масштабованого онлайн-сервісу, що надає можливість здійснювати конвертацію документів з використанням хмарної інфраструктури, забезпечуючи при цьому високу продуктивність, доступність і стабільність функціонування у реальних умовах експлуатації.

1.1.2 Загальні функціональні можливості системи

Онлайн-сервіс конвертації документів призначений для автоматизованої обробки файлів з використанням хмарної інфраструктури та має надавати користувачам такі основні функціональні можливості:

– завантаження документа через вебінтерфейс;
– вибір режиму обробки документа;

– ініціалізація процесу конвертації;
– перегляд статусу виконання завдання;
– отримання готового файлу після завершення обробки.

Система працює без необхідності реєстрації або автентифікації, що спрошує доступ до сервісу та підвищує зручність використання для кінцевого користувача. Весь функціонал доступний без прив’язки до облікового запису, що робить сервіс максимально доступним і швидким у використанні.

1.2 Нефункціональні вимоги

1.2.1 Інтерфейс користувача

Інтерфейс користувача системи повинен забезпечувати коректне відображення та функціонування на різних типах пристроїв, включаючи настільки комп’ютери, планшети та мобільні телефони. Застосунок повинен бути адаптованим з урахуванням різних роздільних здатностей екрану та розмірів вікон браузера. Інтерфейс має бути інтуїтивно зрозумілим, з логічною структурою навігації, що дозволяє користувачу легко та швидко виконувати необхідні операції. Дизайн системи повинен мінімізувати кількість дій, необхідних для досягнення поставлений цілей.

1.2.2 Підтримка браузерів

Система повинна забезпечувати конкретну роботу в актуальних версіях основних веббраузерів:

– Mozilla Firefox;
– Google Chrome;
– Microsoft Edge;
– Safari.

1.2.3 Вимоги до продуктивності

Інтерфейс головної сторінки системи повинен завантажуватися і реагувати на дії користувача не довше ніж за 2 секунди при нормальних умовах роботи та стабільному інтернет-з’єднанні.

Максимальний розмір оброблюваного файлу не повинен перевищувати 50 МБ.

Час обробки одного файлу не має перевищувати 30 секунд.

Система повинна підтримувати одночасно обробку 20-30 запитів без суттєвого зниження продуктивності.

1.2.4 Вимоги до безпеки

Система забезпечує захист завантаження файлів від несанкціонованого доступу шляхом унікального ідентифікування та збереження. Дані передаються зашифрованим каналом, що мінімізує ризик перехоплення.

Передбачено контроль розміру завантажуваних файлів для запобігання перенавантаженню системи.

Логування подій здійснюється для забезпечення контролю роботи системи.

Розгортання системи передбачає застосування заходів безпеки для захисту від несанкціонованого доступу та втручання.

1.3 Опис предметної області

Предметною областю даного дослідження є обробка та управлінням PDF-документами в хмарному середовищі. Формат PDF (Portable Document Format) широко використовується для зберігання, представлення та архівування документів завдяки своїй платформонезалежності, підтримці

текстового і графічного контенту, вбудованих шрифтів та можливості накладання обмежень на редагування [1]. З розвитком електронного документообігу в світі, бізнесі та інших сферах виникає потреба в ефективних інструментах для роботи з PDF, що дозволяє конвертувати, редагувати, та організовувати документи без необхідності встановлення локального програмного забезпечення.

Сучасні сервіси часто обмежені в функціональності або потребують платної підписки, що створює бар'єр для широкого використання [2]. Це зумовлює актуальність розробки доступних та простих у користуванні онлайн-систем для роботи з PDF-документами.

1.4 Опис системи

Розроблено прототип онлайн-сервісу для базової обробки PDF-документів через вебінтерфейс. Система дозволяє користувачам завантажувати PDF-файли та виконувати основні операції без встановлення додаткового програмного забезпечення. Основні функціональні можливості прототипу:

- конвертація першої сторінки PDF у зображення формату PNG для подальшого використання в презентаціях або системах попереднього перегляду;
- злиття кількох PDF-файлів в один документ, що спрощує підготовку звітів або навчальних матеріалів;
- розбиття PDF-документів на окремі сторінки або діапазони сторінок для вибіркового виділення контенту.

Інтерфейс користувача включає завантаження файлів, вибір операцій та завантаження оброблених документів. Система орієнтована на студентів, викладачів та працівників малого та середнього бізнесу, які потребують швидкого і зручного інструменту для базового редагування PDF.

1.5 Порівняння з подібними системами

1.5.1 Аналіз провідних хмарних платформ

Для розробки онлайн-сервісу з обробки PDF-документів вибір надійної хмарної платформи є критично важливим. Серед найбільш поширених платформ варто виділити Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure (див. табл. 1.1).

AWS є провідним постачальником хмарних послуг, що пропонує широкий спектр сервісів для обробки даних, масштабування та безпеки [3]. GCP забезпечує потужні аналітичні інструменти, глибоку інтеграцію з технологіями штучного інтелекту та машинного навчання, що є перевагою для сучасних вебдодатків [4]. Microsoft Azure відомий глибокою інтеграцією з корпоративними продуктами Microsoft, а також широкими можливостями для масштабування та адміністрування [5].

Таблиця 1.1 – Порівняльна характеристика хмарних платформ

Платформа	**Основні переваги**	**Основні недоліки**
AWS	Швидкий спектр сервісів, висока надійність	Високі витрати для невеликих проєктів
Google Cloud	Потужні аналітичні інструменти, AI/ML	Менша кількість сервісів порівняно з AWS
Microsoft Azure	Інтеграція з продуктами Microsoft	Складність освоєння для початківців

Даний аналіз базується на дослідженнях, які висвітлюють фактори впливу хмарних технологій на бізнес [2]. Автори систематично розглядають особливості прийняття хмарних рішень у бізнес-середовищі, що дає змогу краще зрозуміти важливість вибору відповідної платформи.

1.5.2 Порівняння з аналогами для конвертації документів

Існують різні сервіси, що реалізують функції конвертації PDF-документів. Серед найбільш відомих – Simple PDF, PDF Merge та Adobe Acrobat Online (див. табл. 1.2).

Таблиця 1.2 – Порівняння сервісів онлайн-конвертації PDF-документів

Сервіс	Переваги	Недоліки	Бізнес-модель
Simple PDF	Простота, безкоштовний базовий функціонал	Лімітовані безкоштовні функції	Freemium
PDF Merge	Інтуїтивність, швидкість	Обмежені можливості редагування	Безкоштовний / реклама
Adobe Acrobat Online	Широкий функціонал, авторитет бренду	Необхідність реєстрації, обмеження безкоштовної версії	Freemium

Примітка. Термін «Freemium» означає бізнес-модель, за якою базові послуги надаються безкоштовно, тоді як розширений функціонал доступний за плату.

Simple PDF надає базові можливості конвертації та розбиття файлів, однак має обмеження щодо кількості безкоштовних операцій [6]. PDF Merge спеціалізується на злитті документів, але не підтримує пакетне редагування чи конвертацію сторінок у зображення [7]. Adobe Acrobat Online пропонує розширений набір функцій але для обробки документів, включаючи злиття, редагування та конвертацію, проте для повного доступу до сервісів потрібна платна підписка [8].

1.5.3 Унікальність та переваги розробленої системи

Розроблений сервіс має суттєві переваги у порівнянні з існуючими аналогами. Він передбачає повністю безкоштовний доступ, що робить його

доступним для широкої аудиторії, на відміну від більшості конкурентів, які пропонують платні підписки. Система підтримує пакетне редагування, що дозволяє одночасно застосовувати однакові операції до кількох документів. Вебінтерфейс розроблено з урахуванням простоти та інтуїтивності, що забезпечує зручність роботи на будь-яких пристроях без потреби встановлення додаткового програмного забезпечення. Також система інтегрована з хмарними сховищами, що покращує процес збереження та доступу до файлів.

Недоліками є обмежений функціонал на стадії прототипу, що потребує подальшого розвитку.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Використання UML під час розробки системи

UML (Unified Modeling Language) є стандартною мовою моделювання, яка широко застосовується для візуалізації, специфікації, конструювання та документування компонентів програмного забезпечення. Його використання є ключовим під час проєктування складних програмних систем, оскільки UML забезпечує єдину мову комунікації між аналітиками, розробниками, тестувальниками та іншими учасниками проєкту.

UML-моделі дозволяють створити формалізоване уявлення про структуру і поведінку майбутньої системи ще до початку її реалізації. Завдяки наявності різних типів діаграм (діаграм випадків використання, класів, діяльності, послідовностей тощо), розробники можуть чітко окреслити функціональні можливості системи, встановити зв’язки між компонентами, визначити їхню відповідальність та взаємодію.

У процесі розробки даного онлайн-сервісу UML застосовувався для побудови діаграм, які описують архітектуру компонентів, взаємодію користувача із системою, внутрішні бізнес-процеси та логіку обробки запитів. Це дозволило підвищити якість технічного проєктування, зменшити ризики на етапі реалізації та полегшити подальший супровід проєкту.

Значення UML як інструменту ефективного моделювання підтверджується численними науковими і навчальними джерелами. Зокрема, у посібнику К. М. Лавріщевої зазначено, що застосування UML сприяє стандартизації процесів аналізу та проєктування програмних систем, дозволяє зменшити кількість помилок при розробці та забезпечує ефективну комунікацію між усіма учасниками життєвого циклу програмного забезпечення [9].

2.2 Діаграма варіантів використання

Діаграма варіантів використання (англ. Use Case Diagram) є одним із ключових інструментів методології UML, що дозволяє моделювати взаємодію користувачів або зовнішніх систем з програмним продуктом. Основною метою цієї діаграми є формалізації вимог до системи, що виражені через типові сценарії її використання. Це дає змогу розробникам, замовникам і тестувальникам однаково інтерпретувати функціональні можливості системи.

На діаграмі зображуються актори (користувачі, адміністратори, зовнішні сервіси) та варіанти використання, які відображають окремі функції або сценарії взаємодії. Такий підхід дозволяє виявляти як основні, так і допоміжні процеси, що відбуваються у межах системи, а також підкреслити взаємозв’язки між ними.

На рисунку 2.1 представлено базову діаграму варіантів використання розробленого онлайн-сервісу для конвертації документів.

У системі визначено два основних актори: «Кінцевий користувач» та «Системний адміністратор». Перший взаємодіє із сервісом через вебінтерфейс. Основні дії, доступні користувачу, включають завантаження PDF-файлів, ініціацію процесу конвертації, обробку документів у пакетному режимі, перегляд статусу завдання, отримання результату та його збереження. У рамках конвертації користувач може скористатися основними можливостями, такими як перетворення у PNG, об’єднання кількох файлів або розділення одного документа на декілька.

«Системний адміністратор», у свою чергу, має доступ до функцій, що забезпечують стабільну роботу сервісу. Зокрема, він може переглядати журнал подій, керувати чергою завдань та за потреби видаляти окремі задачі з системи.

Ключові варіанти використання охоплюють усі основні сценарії взаємодії з системою. Завантаження PDF-файлу є точкою входу в процес, після чого можливе перетворення документа відповідно до обраного режиму.

Статус кожного завдання можна перевірити в режимі реального часу, після чого, користувач отримує результат та має змогу його зберегти.



Рисунок 2.1 – Діаграма варіантів використання

Таким чином, діаграма варіантів використання дозволяє систематизувати функціональні можливості онлайн-сервісу, чітко розмежувати права доступу між ролями та забезпечити узгодження логіки роботи системи на етапі розробки.

2.2.1 Опис варіантів використання

Прецедент «Конвертація документів» (загальний прецедент для групування дій над PDF-файлами).

Призначення: даний варіант використання дозволяє користувачу ініціювати процес конвертації PDF-документа в один або кілька вихідних форматів. Сценарій охоплює як одиночну, так і пакетну обробку файлів.

Основний потік подій: користувач завантажує PDF-файл через вебінтерфейс, після чого обирає тип конвертації – наприклад, об'єднання кількох файлів в один. Система приймає завдання, передає його в чергу на обробку, а після завершення формує посилання для отримання результату.

Альтернативний потік подій: якщо файл пошкоджений або не є PDF-документом, система повідомляє про помилку та не додає його до черги.

Прецедент «Пакетна обробка» (прецедент, який має зв'язко "include" з базовим прецедентом «Конвертація документів»).

Призначення: надає можливість користувачу обробляти одразу декілька PDF-файлів для їх об'єднання.

Основний потік подій: користувач завантажує декілька файлів, система групує їх в одну задачу, застосовує вибрану операцію, обробляє та формує результати про завершенні черги.

Прецедент «Перегляд статусу завдання» (прецедент, що має зв'язок "extend" із прецедентом «Конвертація документів»).

Призначення: дозволяє користувачу перевірити стан виконання конкретного завдання: у черзі, обробляється, завершено, помилка тощо.

Основний потік подій: користувач отримує контексті повідомлення на головному екрані у відповідь на дії: додано завдання до черги, завершено обробку, помилка під час виконання, некоректний набір файлів.

Передумова: користувач вже створив хоча б одне завдання на конвертацію.

Виняткова ситуація: якщо завдання не знайдено або термін зберігання минув, система виводить повідомлення «Завдання не знайдено або завершено».

Прецедент «Отримати результат» (розширює прецедент «Перегляд статусу завдання»).

Призначення: дозволяє користувачу завантажити готовий файл після завершення конвертації.

Основний потік подій: коли статус завдання змінюється на «Завершено», користувач отримує посилання на готовий документ, яке дозволяє переглянути його у браузері або завантажити на власний пристрій.

Передумова: завдання повинно бути завершено.

Виняткова ситуація: файл недоступний через внутрішню помилку – система виводить повідомлення «Помилка при завантаженні результату».

Прецедент «Зберегти результат» (розширює прецедент «Отримати результат»).

Призначення: надає користувачу можливість зберегти отриманий результат конвертації на свій пристрій.

Основний потік подій: після отримання посилання на готовий документ у хмарному сховищі користувач натискає на посилання, система відкриває доступ до файлу, що дозволяє зберегти його на пристрій.

Передумова: користувач отримав файл.

Прецедент «Перегляд журналу подій» (доступний лише адміністратору).

Призначення: надає адміністратору можливість відслідковувати дії системи, діагностувати збої та виявляти аномалії в роботі мікросервісів.

Основний потік подій: через панель керування адміністратор відкриває журнал, де переглядає інформацію про завдання, сервіси, час відгуку та інші показними системи.

Прецедент «Керування чергою завдань» (доступний лише адміністратору).

Призначення: дозволяє адміністратору бачити чергу завдань, зупиняти його або видаляти задачі, які не виконуються через помилки або зупинку, а також вручну ініціює обробку.

Основний потік подій: адміністратор відкриває панель черги, переглядає статуси, натискає дії на відповідні задачі.

Передумова: адміністратор має бути авторизованим через спеціальний інтерфейс.

Прецедент «Видалення завдання» (прецедент адміністратора, який може виконуватися автономно або як частина керування чергою).

Призначення: дозволяє адміністратору видаляти непотрібні або застарілі задачі із системи.

Основний потік подій: адміністратор натискає «Видалити» поруч із обраним завданням. Система запитує підтвердження, після чого виконує видалення та оновлює список задач.

2.3 Діаграма послідовності

Діаграма послідовностей (англ. Sequence Diagram) – це від діаграм UML, який використовується для моделювання динамічної поведінки системи, демонструючи порядок обміну повідомлення між об’єктами або компонентами в межах певного сценарію. Вона відображає послідовність взаємодії у часі, допомагаючи зрозуміти, як саме відбувається обробка запитів та виконання операцій у системі [10].

На рисунку 2.2 представлена діаграма послідовності, що ілюструє основний сценарій обробки файлу у сервісі онлайн конвертації PDF-документів. Взаємодія починається з ініціації користувачем завантаження файлу через API Gateway, який приймає вхідні дані та передає їх до системи черг RabbitMQ. Черга забезпечує асинхронну постановку завдання, яке надалі отримує робочий вузол Worker для виконання операцій конвертації та обробки документа за допомогою PDFLib і Ghostscript.

Діаграма демонструє послідовність викликів методів між компонентами: від завантаження та постановки завдання у чергу, через обробку на робочих вузлах, до збереження результату у сховищі Amazon S3. Завершення обробки супроводжується надсиланням push-повідомлення через

сервіс Pusher, що інформує користувача про готовність результату. Взаємодія відбувається з урахуванням асинхронності, що підвищує продуктивність і відмовостійкість системи.

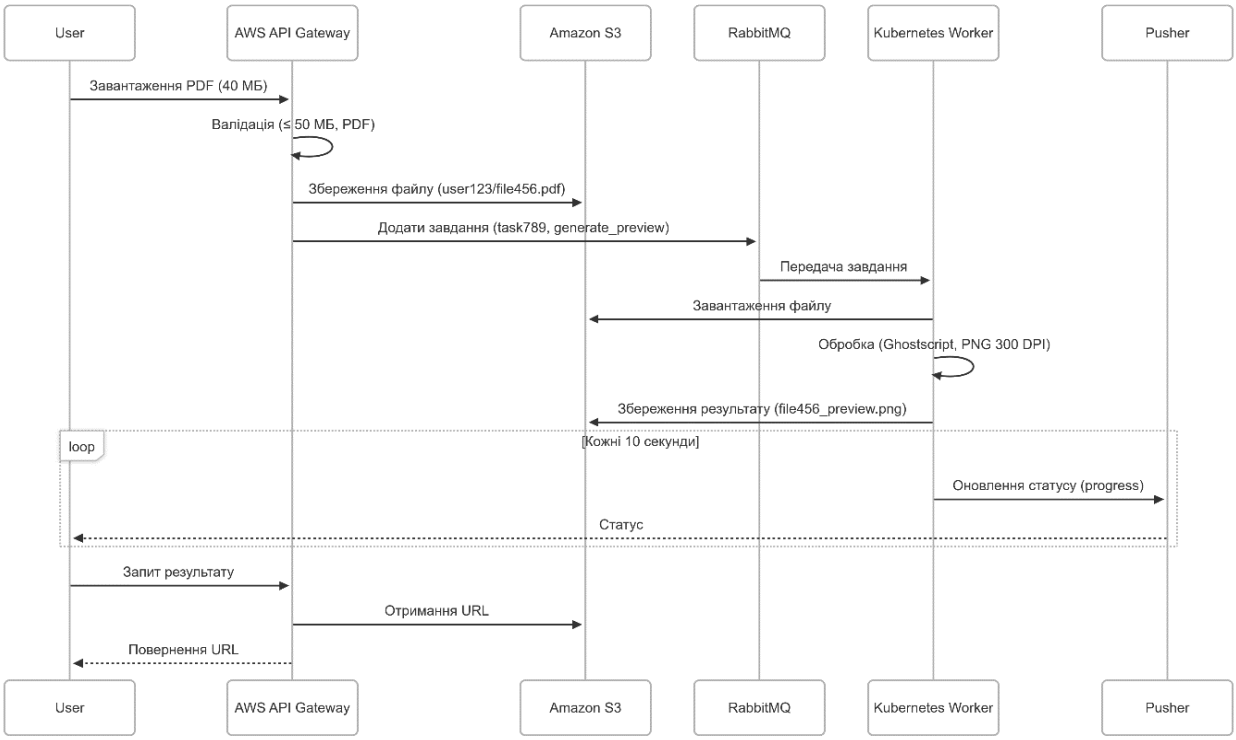


Рисунок 2.2 – Діаграма послідовності

Питання архітектурної організації цих компонентів буде детально розглянуто у розділі, де представлено діаграму компонентів.

Таким чином, діаграма послідовності є візуальним підтвердженням архітектурних рішень, описаних у попередніх розділах, і служить інструментом для аналізу і оптимізації процесів у системі, дозволяючи чітко відстежувати кожен крок від запиту користувача до отримання готового результату.

2.4 Діаграма класів

Діаграма класів є однією з найважливіших діаграм в UML, що використовується для статичного моделювання структури об’єктно-

орієнтованих систем. Вона показує класи системи, їх атрибути, методи та взаємозв’язок між ними [10].

Клас в об’єктно-орієнтованому програмуванні представляє собою шаблон або «креслення» для створення об’єктів, що описує їх структуру (атрибути) та поведінку (методи). Класи забезпечують інкапсуляцію даних та функціональність, дозволяючи створювати модульні та повторно використовувані компоненти системи. На рисунку 2.3 представлено діаграму класів.

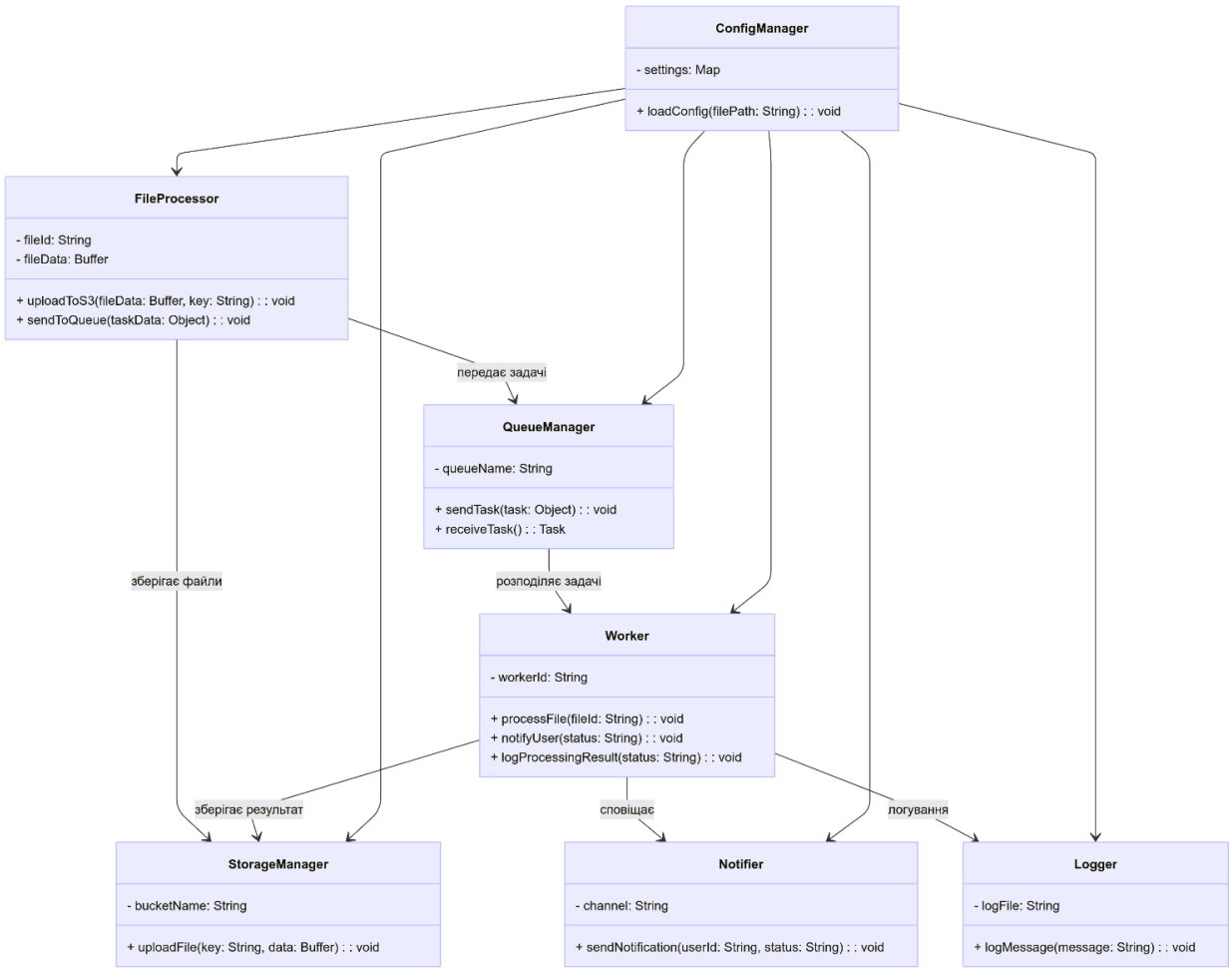


Рисунок 2.3 – Діаграма класів

Наведемо опис основних компонентів системи, які відображені на діаграмі класів.

FileProcessor – головний клас, що координує процес завантаження та первинної обробки файлів. Взаємодія з StorageManager для збереження файлів

у хмарному сховищі та з QueueManager для передачі завдань на асинхронну обробку.

StorageManager – відповідає за роботу з Amazon S3, включаючи завантаження файлів та отримання посилань для доступу. Підтримує версіонування документів для забезпечення цілісності даних.

QueueManager – керує чергою завдань через RabbitMQ, забезпечуючи надійну асинхронну обробку з можливістю повторного виконання при збоях.

Worker – виконує безпосередню обробку документів, включаючи генерацію зображень та конвертацію документів. Може масштабуватися горизонтально для обробки великої кількості файлів.

ConfigManager та **Logger** – допоміжні класи для централізованого управління конфігурацією системи та ведення журналу подій відповідно.

Notifier – забезпечує відправку push-повідомлень користувачам про статус обробки документів через сервіс Pusher.

Така архітектура забезпечує модульність системи, дозволяє ефективно масштабувати окремі компоненти та підтримує принципи мікросервісного підходу.

2.5 Діаграма компонентів

Діаграма компонентів (англ. Component Diagram) – це типу діаграми UML, яка моделює фізичні компоненти програмної системи та їхні залежності. Вона дозволяє зобразити архітектурну структуру програмного забезпечення, поділену на логічно ізольовані частини (компоненти), які реалізують певну функціональність. Такі діаграми широко використовуються для моделювання систем з розподіленою архітектурою, особливо при роботі з хмарними технологіями, мікросервісами та повідомленнями [10].

На рисунку 2.4 представлено діаграму компонентів розробленої хмарної онлайн платформи для конвертації документів. Вона відображає основні

компоненти системи, взаємозв’язки між ними, типи сервісів та допоміжні утиліти.

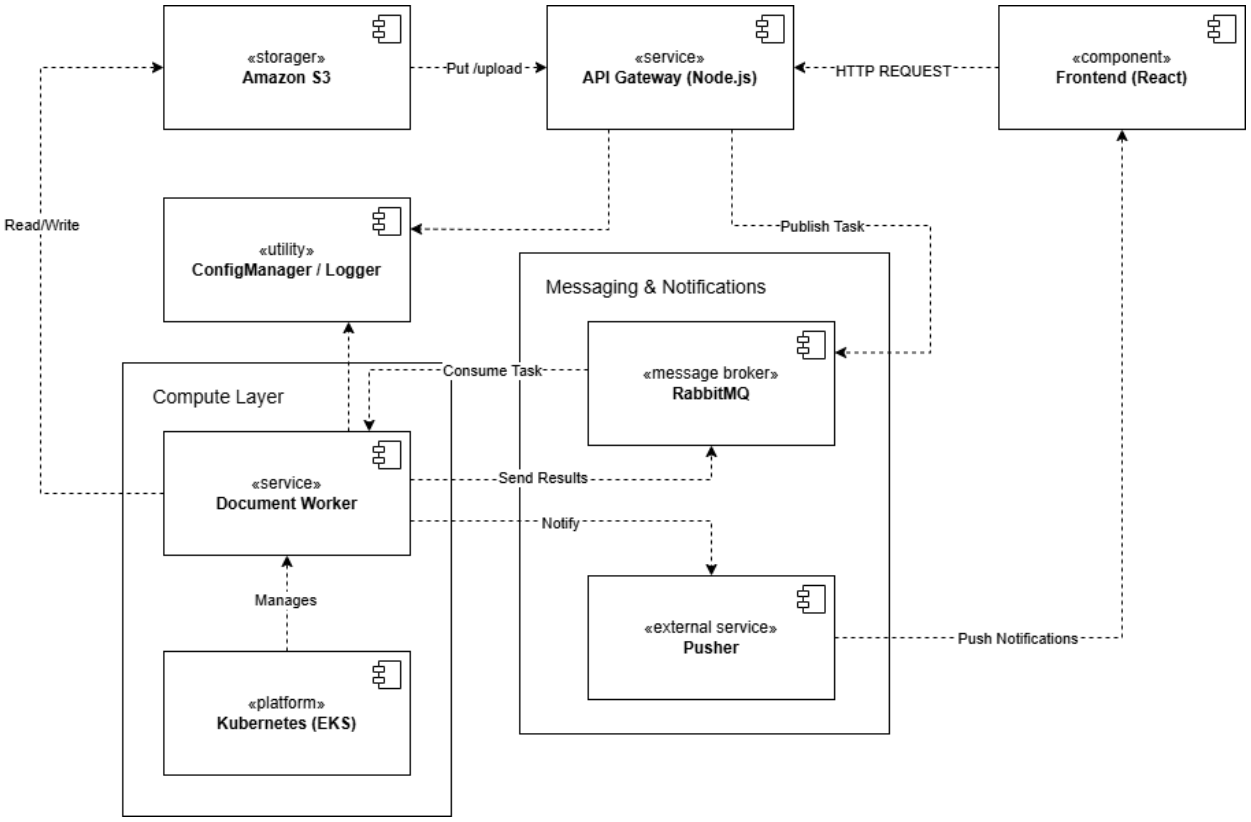


Рисунок 2.4 – Діаграма послідовності

На діаграмі умовно виокремлено два ключових блоки: Compute Layer, де відбувається основна обробка документів, та Messaging & Notifications, що забезпечують взаємодію між компонентами. Варто зазначити, що діаграма ілюструє лише основні частини системи й не охоплює всі другорядні залежності та сервіси.

Frontend (React) – інтерфейс користувача, що надсилає HTTP-запити до API Gateway та отримує в реальному часі повідомлення про статус виконання завдань.

API Gateway (Node.js) – шлюз, який обробляє запити клієнтів, завантажує файли в S3, публікує завдання в чергу RabbitMQ, повертає клієнту статуси.

RabbitMQ – брокер повідомлень, що виступає як посередник між компонентами, забезпечуючи асинхронну передачу завдань та результатів.

Document Worker – обчислювальний компонент, що отримує завдання з RabbitMQ, виконує конвертацію та зберігає результат у S3. Після завершення надсилає результати назад у чергу та повідомляє Pusher.

Pusher – сервіс реального часу для надсилання повідомлень клієнту щодо статусу обробки документів.

Amazon S3 – об’єктне сховище для завантажених користувачем документів та результатів їхньої конвертації.

Kubernetes (Amazon EKS) – платформа для оркестрації контейнеризованих обчислювальних сервісів, зокрема Document Worker.

ConfigManager / Logger – допоміжні утиліти для централізованого управління конфігураціями та ведення журналів логування.

Ця діаграма дозволяє наочно продемонструвати структуру взаємодії компонентів системи, вказати ці сервіси розгорнуті в хмарі, які відповідають за обчислення, зберігання, комунікацію та підтримку.

2.6 Програмний інтерфейс API

Програмний інтерфейс API є ключовим елементом взаємодії клієнтських додатків із системою обробки документів. Його реалізовано з використанням AWS API Gateway та REST-протоколу, що забезпечує надійність, масштабованість і підтримку асинхронної обробки запитів. API оптимізовано для роботи з великими файлами обсягом до 50 МБ.

Функціональність API охоплює три основні категорії операцій: завантаження та моніторинг, обробку документів і системне управління.

У межах першої категорії реалізовано можливості завантаження файлів, перевірки статусу їх обробки, отримання готових результатів та попереднього перегляду.

Обробка документів включає створення попереднього перегляду, об’єднання, розділення та видалення PDF-файлів.

Системне управління охоплює доступ до конфігурацій, перегляд журналів, діагностику помилок і можливість примусово зупиняти процеси обробки.

Детальна специфікація API представлена в таблиці 2.1.

Таблиця 2.1 – Порівняльна характеристика хмарних платформ

Категорія	**Ендпоінт**	**Метод**	**Призначення**
Завантаження/ моніторинг	/upload	POST	Завантаження PDF-документу
	/status/:fileId	GET	Отримання статусу обробки
	/result/:fileId	GET	Отримання посилання на файл
	/preview/:fileId	Get	Отримання попереднього перегляду
Обробка документів	/generate-preview/:fileId	POST	Створення зображення першої сторінки
	/merge/:fields	POST	Об’єднання PDF-файлів
	/split/:fileId	POST	Розділення PDF на сторінки
	/files/:fileId	DELETE	Видалення файлу
Системне управління	/config	GET/PUT	Отримання/оновлення конфігурації
	/cancel/:fileId	POST	Скасування обробки файлу
	/errors/:fileId	GET	Перегляд історії помилок
	/logs	GET	Отримання системних журналів
	/cancel/:fileId	POST	Скасування обробки файлу

Ендпоінт /upload ініціює збереження файлу в Amazon S3 через StorageManager та постановку завдання в RabbitMQ через QueueManager. Отримання статусу (/status/:fileId}) реалізується з використанням Notifier та Pusher.

Операції обробки документів (наприклад /merge/fields}) виконуються через компонент Worker, який використовує PDFLib для генерації попереднього перегляду, а також Ghostscript для об’єднання та розділення PDF-файлів.

Системні точки доступу взаємодіють з QueueManager, ConfigManager, Logger, що забезпечує повний контроль над системою.

Діаграма потоків даних продемонстрована на рисунку 2.5.

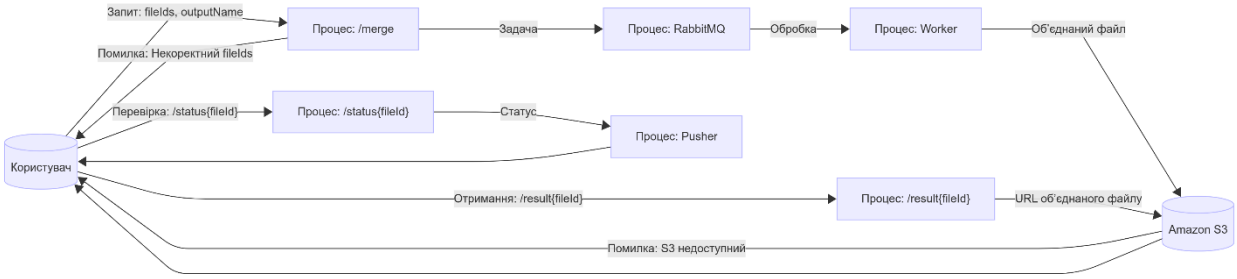


Рисунок 2.5 – Діаграма потоків даних процесу об’єднання файлів

Діаграма ілюструє взаємодію ендпоінтів з компонентами системи під час об’єднання документів, включаючи передачу файлів у S3, постановку задач у RabbitMQ та обробку через Worker.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Реалізація програмного забезпечення

Цей розділ описує практичну реалізацію компонентів розробленого онлайн-сервісу для конвертації, злиття та розбиття PDF-документів. Застосовано мікросервісну архітектуру з використанням хмарних технологій, що забезпечує гнучкість, масштабування та ефективну обробку завдань. Основна увага приділяється побудові API Gateway, системі асинхронної обробки, а також інтеграції з хмарними сервісами для зберігання та оркестрації.

Реалізація базується на технологічному стеку, який включає Node.js із фреймворком Express для API Gateway, RabbitMQ для асинхронної обробки, PDFLib і Ghostscript для трансформації PDF, React для фронтенду, а також теоретичне використання Amazon EKS (Kubernetes) для оркестрації та Amazon S3 для зберігання файлів. Локальна розробка виконана з використанням Docker Compose, що імітує розподілену інфраструктуру через обмеження доступу до AWS Educate.

Особливістю реалізації є поділ відповідальностей між компонентами: API Gateway приймає запити, перевіряє їх коректність і передає завдання у чергу RabbitMQ, тоді як сервіси обробки асинхронно обробляють файли, зберігаючи результати у локальному сховищі (./storage) або в S3. Такий підхід дозволяє уникнути затримок для користувача та забезпечує горизонтальне масштабування.

Фронтенд на React забезпечує зручний інтерфейс і доступ до основних операцій (завантаження, конвертація, збереження). Механізм сповіщення реалізовано за допомогою сервісу Pusher (WebSocket), що забезпечує миттєве оновлення статусу завдань у клієнтському інтерфейсі. У локальному режимі, за відсутності доступу до хмарних сервісів, застосовувалось періодичне опитування через ендпоінт /task/:taskId як тимчасову альтернативу.

Ключовими аспектами реалізації є підтримка продуктивності (обробка 50 МБ файлів ≤ 60 секунд при ≥ 20 запитах), логування через Winston для моніторингу операцій і планування інтеграції з хмарними сервісами для забезпечення SLA 99,9%. Усі компоненти протестовано локально, із акцентом на асинхронність і відмовостійкість, що робить систему готовою для подальшого розгортання в реальному середовищі.

3.1.1 Реалізація API Gateway

У системі онлайн-конвертації документів компонент API Gateway виконує роль центрального шлюзу, який забезпечує єдину точку входу для всіх HTTP-запитів з боку клієнтської частини. Його основні функції включають прийом запитів, валідацію вхідних даних, збереження файлів, постановку завдань у чергу RabbitMQ та координацію обміну інформацією між клієнтом і внутрішніми сервісами.

API Gateway реалізований на основі середовища Node.js з використанням фреймворку Express, що забезпечує ефективну обробку маршрутизованих запитів. Для обробки вхідних файлів застосовується бібліотека Multer у режимі memoryStorage, що дозволяє зчитувати файли безпосередньо в оперативну пам’ять. Асинхронну взаємодію з бекендом реалізовано за допомогою RabbitMQ, та продемонстровано на рисунку 3.1.

При находженні POST-запиту на ендпоінт /upload API Gateway спочатку перевіряє тип файлу – допускаються лише PDF, а також контролює максимальний розмір файлу (до 50 МБ). Після перевірки файл тимчасово зберігається у локальному каталозі ./storage. Хоча для зберігання файлів передбачена можливість робити з Amazon S3 через AWS SKD, що було протестовано в рамках середовища AWS Educate Starter Account, основне збереження наразі відбувається локально.

Далі створюється унікальний ідентифікатор taskId (UUID), після чого завдання з параметрами файлу передається у чергу pdf-tasks (RabbitMQ). Після

обробки спеціальним обчислювальним компонентом, результати публікуються в чергу pdf-results, звідки API Gateway отримує їх, оновлює статус у колекції taskStatuses (структура типу Map) і надає клієнту можливість отримувати результат за запитом до ендпоінту /task/:taskId.

```
const amqp = require('amqplib');
const uuid = require('uuid');

async function connect() {
  const connection = await amqp.connect(`amqp://${process.env.RABBITMQ_USER}:${process.env.RABBITMQ_PASSWORD}@${process.env.RABBITMQ_HOST}`);
  const channel = await connection.createChannel();
  await channel.assertQueue('pdf-tasks', { durable: true });
  await channel.assertQueue('pdf-results', { durable: true });
  return { connection, channel };
}

async function sendTask(channel, fileKey, taskType) {
  const taskId = uuid.v4();
  const task = { id: taskId, fileKey, type: taskType };
  channel.sendToQueue('pdf-tasks', Buffer.from(JSON.stringify(task)), { persistent: true });
  return taskId;
}

module.exports = { connect, sendTask };
```

Рисунок 3.1 – Фрагмент коду взаємодії з RabbitMQ

Для моніторингу роботи шлюзу застосовується бібліотека Winston (див. рис. 3.2), яка збирає інформацію про ключові події: надходження запитів, помилки перевірки, відправку завдань у RabbitMQ, отримання результатів та виняткові ситуації. Логи зберігаються у файлах combined.log і error.log та виводяться у консоль для зручності налагодження.

```
const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    new winston.transports.File({ filename: 'error.log', level: 'error' }),
    new winston.transports.File({ filename: 'combined.log' }),
    new winston.transports.Console()
  ]
});

module.exports = logger;
```

Рисунок 3.2 – Фрагмент коду налаштування логування в API Gateway за допомогою бібліотеки Winston

В межах AWS Educate Starter Account вдалося успішно реалізувати інтеграцію з Amazon S3. Проте через обмеження облікового запису – відсутність доступу до IAM, обмеження на запуск EKS, а також неможливість використання сервісів моніторингу і повідомлень (CloudWatch, SNS, SQS) – повноцінна масштабована інтеграція та моніторинг залишається на рівні концепту.

3.1.2 Реалізація компонентів асинхронної обробки документів

Компоненти асинхронної обробки документів реалізовано у вигляді фонових обробників (worker-процесів), які відповідають за виконання завдань із черги RabbitMQ. Завдяки асинхронній моделі обробки зменшуються затримки у взаємодії з користувачем, підвищується продуктивність системи та досягається горизонтальна масштабованість за рахунок розгортання додаткових обробників.

Для реалізації PDF-операцій було використано дві ключові бібліотеки: PDFLib та Ghostscript. Перша дозволяє ефективно зливати та розділяти PDF-документи на рівні JavaScript коду без необхідності сторонніх залежностей. У свою чергу, Ghostscript застосовується для високоякісної конвертації PDF у PNG-формат із роздільною здатністю 300 DPI. Такий підхід забезпечує прийнятну швидкість обробки та водночас високу якість вихідних зображень.

Кожен worker встановлює з'єднання з RabbitMQ за допомогою бібліотеки amqplib та підписується на чергу pdf-tasks (дис. рис. 3.3).

```
const amqp = require('amqplib');

async function startWorker() {
  const connection = await amqp.connect('amqp://localhost');
  const channel = await connection.createChannel();
  await channel.assertQueue('pdf-tasks');
  await channel.assertQueue('pdf-results');

  channel.consume('pdf-tasks', async msg => {
    if (!msg) return;
    const { taskId, fileKey, operation } = JSON.parse(msg.content.toString());
    try {
      const result = await processDocument(taskId, fileKey, operation);
      channel.sendToQueue('pdf-results', Buffer.from(JSON.stringify({ taskId,
status: 'success', result })));
      channel.ack(msg);
    } catch (error) {
      channel.sendToQueue('pdf-results', Buffer.from(JSON.stringify({ taskId,
status: 'error', error: error.message })));
      channel.ack(msg);
    }
  });
}
```

Рисунок 3.3 – Worker-процес, що підключається до RabbitMQ, обробляє повідомлення та надсилає результат

Після отримання повідомлення, що містить taskId, fileKey та тип операції (convert, merge, split), обробник завантажує відповідний файл із обраного сховища, виконує потрібну дію та надсилає результат (або повідомлення про помилку) у чергу pdf-results. Успішність обробки підтверджується за допомогою ack.

Для зберігання вхідних та вихідних файлів використовується хмарне сховище Amazon S3. Організація структури даних на S3 здійснюється логічне розбиття на папки (префікси), що дозволяє ефективно управляти файлами та їх доступом (див. рис. 3.4).

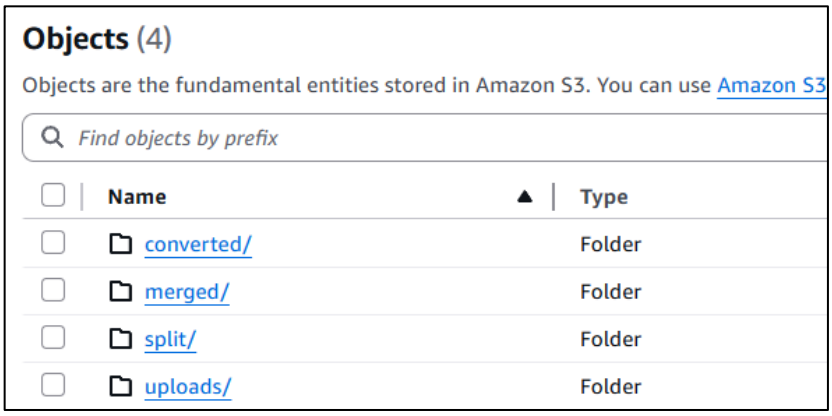


Рисунок 3.4 – Ієрархія папок у S3 для зберігання вхідних та оброблених файлів

Наприклад, для завантажених користувачем PDF-файлів створюється папка uploads/, конвертовані зображення зберігаються у папці converted/ злиті документи – у папці merged/, а розділені – у папці split/. Така ієрархія дозволяє швидко ідентифікувати та знаходити потрібні файли, а також забезпечує чистоту та порядок у сховищі. Реалізація функції представлена на рисунку 3.5.

Взаємодія з S3 реалізована через офіційний AWS SDK (рис 3.6), що дає змогу не тільки завантажувати файли, але й формувати підписані URL-адреси для тимчасового доступу до оброблених документів. Це важливо для безпечного поширення результатів користувачам без потреби відкривати загальний доступ до хмарного сховища. Завдяки такому підходу система зберігає високий рівень безпеки й одночасно гарантує зручність використання.

```
const PDFLib = require('pdf-lib');
const { exec } = require('child_process');

async function mergePDFs(files) {
  const mergedPdf = await PDFLib.PDFDocument.create();
  for (const file of files) {
    const pdfBytes = await fetchFile(file);
    const pdf = await PDFLib.PDFDocument.load(pdfBytes);
    const copiedPages = await mergedPdf.copyPages(pdf, pdf.getPageIndices());
    copiedPages.forEach(page => mergedPdf.addPage(page));
  }
  return await mergedPdf.save();
}

const convertPDFtoPNG = (inputPath, outputPath) =>
  new Promise((res, rej) => exec(`gs -dNOPAUSE -dBATCH -
sDEVICE=png16m -r300 -sOutputFile=${outputPath} ${inputPath}`, err => err
? rej(err) : res(outputPath)));
```

Рисунок 3.5 – Функції для злиття PDF, та конвертації PDF у PNG

```
const AWS = require('aws-sdk');
const s3 = new AWS.S3();

async function uploadFile(bucket, key, body) {
  await s3.putObject({
    Bucket: bucket,
    Key: key,
    Body: body,
  }).promise();
}

function getSignedUrl(bucket, key, expiresSeconds = 3600) {
  return s3.getSignedUrl('getObject', {
    Bucket: bucket,
    Key: key,
    Expires: expiresSeconds,
  });
}
```

Рисунок 3.6 – Робота з Amazon S3: завантаження файлів і генерація підписах URL

3.1.3 Реалізація вебінтерфейсу

Вебзастосунок виступає центральним компонентом системи, який забезпечує користувачу зручний і інтуїтивний доступ до операцій обробки PDF-документів.

Для забезпечення зручної та ефективної роботи з документами вебінтерфейс реалізовано з акцентом на простоту використання і швидкий доступ до основних функцій. Інтерфейс побудований із застосуванням сучасних технологій, зокрема бібліотеки Material-UI, що забезпечує адаптивний, чистий і зручний дизайн. Це дозволяє швидко завантажувати файли та отримувати оновлення статусу обробки.

Інтерфейс побудовано навколо одного робочого екрану з функціональною областю завантаження файлів, реалізованою за допомогою технології drag-and-drop. Користувач може додавати документ як простим перетягуванням у вікно, так і через стандартний діалог вибору файлів (див. рис. 3.7).

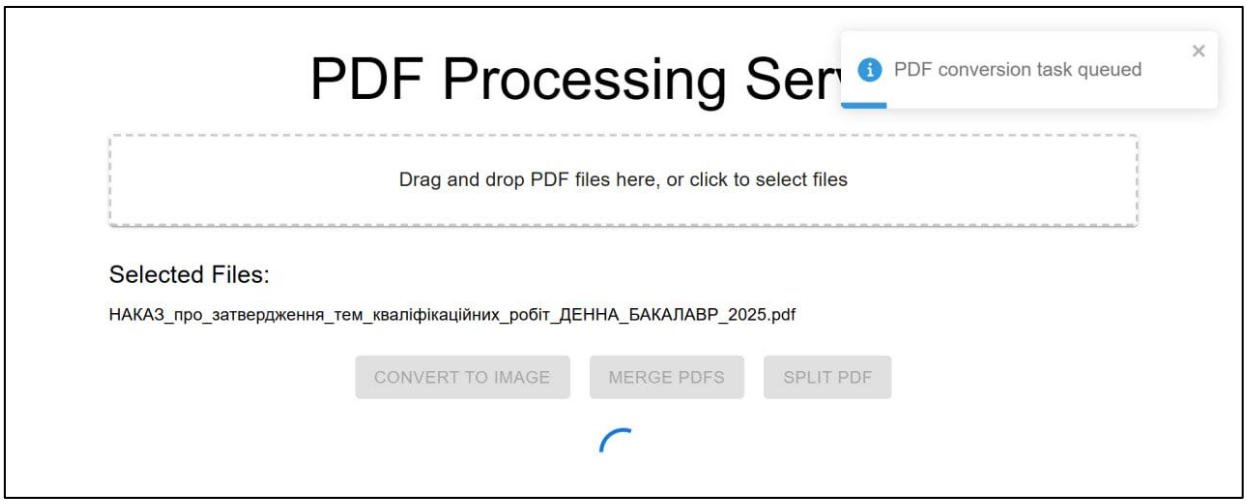


Рисунок 3.7 – Інтерфейс із звантаженим файлом і push-повідомленням про статус обробки

Після додавання файлу користувач має можливість розпочати одну з операцій: конвертацію першої сторінки у PNG, об’єднання документів або розбиття PDF на сторінки. Під час обробки завдання інтерфейс відображає

статус у реальному часі через push-повідомлення, які надходять від сервісу Pusher.

Після успішного завершення обробки результат (PNG-зображення) стає доступним для перегляду і завантаження безпосередньо у вікні вебзастосунку. Це наглядно продемонстровано на рисунку 3.8.

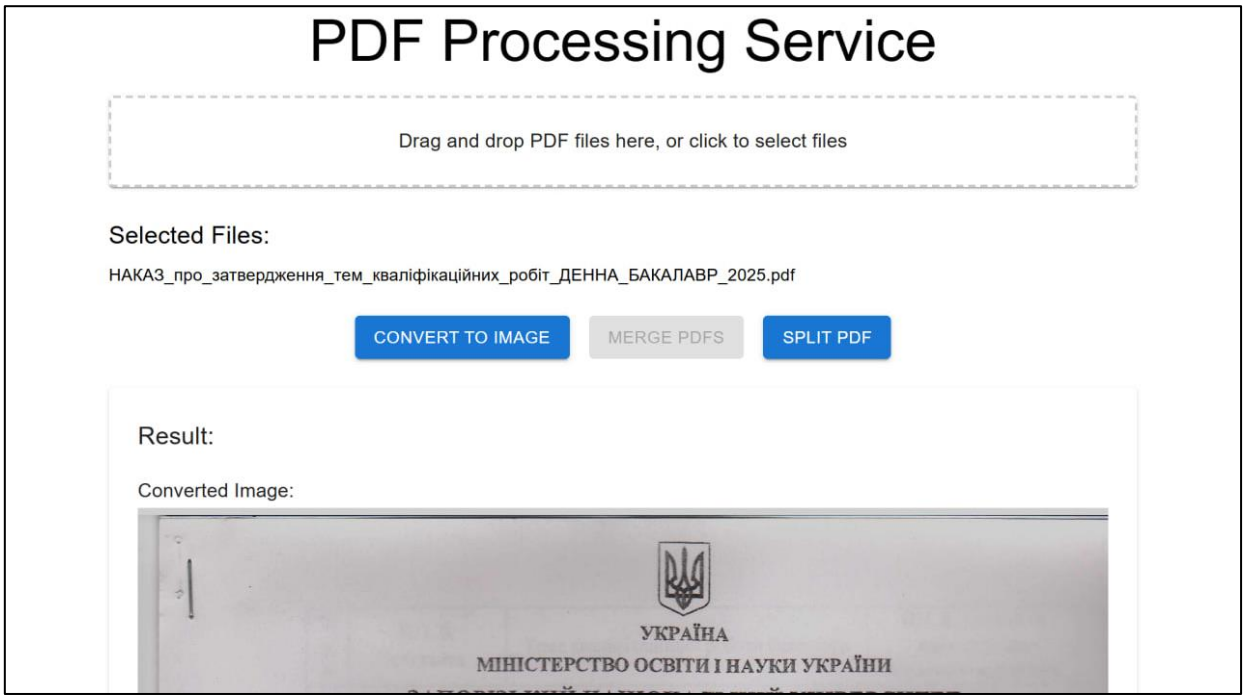


Рисунок 3.8 – Відображення готового PNG-зображення після конвертації

Взаємодія фронтенду з бекендом відбувається через REST API Gateway. Завантаження файлів здійснюється POST-запитами з типом multipart/form-data, а результат обробки зберігається в сховищі S3.

Реалізація на основі React забезпечує компонентний підхід, швидке оновлення віртуального DOM і гнучке управління станом, що є критично важливим для своєчасного відображення змін статусів і результатів.

3.1.4 Реалізація механізму сповіщень

Механізм сповіщень був спроєктований для оновлення статусу завдань у реальному часі за допомогою сервісу Pusher, який працює на основі

WebSocket. Pusher мав забезпечувати миттєву передачу подій (наприклад, “queued”, “processing”, “completed”) від API Gateway до фронтенду, підписаний на каналі через бібліотеку pusher-js, отримував сповіщення та динамічно оновлював інтерфейс. Це знижувало навантаження на сервер і покращувало досвід користувача.

Через обмеження локальної розробки (відсутність доступу до Pusher у тестовому середовищі) було застосовано альтернативний підхід з періодичним HTTP-опитуванням. Фронтенд кожні 10 секунд надсилає GET-запити на ендпоінт /task/:taskId для перевірки статусу завдання. На стороні API Gateway статуси зберігаються у структурі Map (taskStatuses), де ключем є taskId, а значенням – об’єкт зі статусом та повідомленням. Після завершення обробки фоновими процесами (worker), статус оновлюється, і фронтенд отримує актуальну інформацію. Хоча цей спосіб менш ефективний за WebSocket, він дозволяє протестувати основний функціонал.

Завдяки оптимізації роботи API Gateway та швидкому доступу до taskStatuses, затримка оновлення статусу не перевищує 2 секунди, що забезпечує достатньо швидке інформування користувача про хід виконання завдання.

Використання HTTP-опитування як тимчасового рішення демонструє гнучкість системи. Архітектура дозволяє у майбутньому легко змінювати цей механізм на Pusher без суттєвих змін у коді, адже API Gateway і фронтенд працюють із уніфікованим форматом статусів. Це робить системою готовою до масштабування та інтеграції з іншими сервісами.

3.2 Тестування системи

Тестування системи проводилося локально з використанням Docker Compose, що дозволило змоделювати роботу хмарної інфраструктури в умовах обмежень AWS Educate Starter Account. Через обмежений доступ до сервісів AWS, повноцінне розгортання в Amazon Web Services було

неможливим. Для кожного основного компонента – API Gateway, брокер повідомлень RabbitMQ, Worker-процесів та клієнтської частини (фронтенду) – було створено окремі контейнери.

Для модульного тестування застосовувалася бібліотека Jest як для бекенду, так і для фронтенду. Worker-процеси тестувалися за допомогою окремих Node.js-скриптів. Під час перевірки враховувалися нефункціональні вимоги: система мала обробляти файли розміром до 50 МБ за час не більше 30 секунд, а також витримувати навантаження до 20-30 одночасних запитів. Моніторинг і діагностика системи здійснювалися за допомогою бібліотеки Winston, яка забезпечувала логування усіх критичних операцій.

Було протестовано повну логіку API Gateway, включно з обробкою запитів /upload, /status/:fileId та /result/:fileId, перевірку PDF-файлів за розміром і форматом, формування taskId, а також додавання завдань у чергу pdf-tasks в RabbitMQ. У фонових обробників перевірялась коректність виконання завдання із черги, а також передача результату у чергу pdf-results. У середовищі RabbitMQ було перевірено стабільність черг, асинхронну обробку до 30 паралельних завдань, а також надійність збереження завдань завдяки параметру durable. Фронтенд-тестування включало перевірку функціональності drag-and-drop-завантаження файлів, правильне відображення статусу обробки завдань, а також сумісність інтерфейсу з основними браузерами: Chrome, Firefox, Edge і Safari.

Було проведено повноцінне інтеграційне тестування з охопленням усіх етапів роботи системи. Користувач завантажував PDF-файл через форму на клієнтській частині. API Gateway приймав файл, виконував перевірку, зберігав його у локальне сховище або S3, формував унікальний taskId і надсилав завдання у чергу pdf-tasks. Worker-процеси зчитували завдання з черги, виконували необхідні операції (наприклад, конвертацію чи об’єднання), зберігали результат у підкаталогах converted/, після чого повертали статус у чергу pdf-results. API Gateway, отримавши оновлений статус за допомогою періодичного HTTP-запиту до ендпоінта /task/:taskId та показував відповідне повідомлення або посилання на результат.

В рамках тестування перевірялись як одиночна, так і пакетна обробка до п’яти файлів. Також моделювалися типові помилки – завантаження файлів із перевищеним обсягом або некоректною структурою. За результатом тестів система стабільно обробляла 20 паралельних запитів протягом менш ніж 30 секунд, що відповідало технічному завданню.

Логування відіграло ключову роль у діагностиці та відлагодженні. Бібліотека Winston дозволяла зберігати події у файл combined.log (усі події) та error.log (винятки), а також дублювати їх у консоль. У логах фіксуються вхідні запити, створення та виконання завдань, зміна статусу обробки, а також типові помилки, такі як “Invalid PDF structure”, або помилки під час роботи з файловою системою.

На рисунку 3.9 наведено скриншот реального лог-файлу combined.log, де зафіксовано повний цикл обробки запиту: находження на API, передача в RabbitMQ, обробка у Worker-процесі та повернення результату. Логування у форматі JSON дало змогу точно аналізувати час виконання кожного етапу та оперативно виявляти потенційні вузькі місця.



Рисунок 3.9 – Фрагмент логу Winston (combined.log)

Результати тестування підтвердили відповідність системи технічним вимогам. Docker Compose забезпечив успішну симуляцію хмарної інфраструктури. Усі основні компоненти працювали стабільно як окремо, так і в рамках повного робочого циклу. Логування з використанням Winston дозволило виявити помилки на ранньому етапі та спростило налагодження.

У подальших етапах планується розгортання сервісу в повноцінному AWS-середовищі для перевірки масштабованості та надійності в умовах наближених до реального промислового використання.

3.2.1 Аналіз швидкодії та продуктивності

Результати тестування продуктивності показати, що середній час обробки файлів коливається залежно від операції: конвертація PDF у PNG триває приблизно від 10 до 20 секунд, а злиття та розбиття PDF виконуються швидше – в межах 5-10 секунд для файлів розміром від 5 до 50 МБ (див. рис. 3.10). Використання Ghostscript для генерації растрових зображень накладає значне навантаження на систему, що пояснює більшу тривалість конвертації. Система змогла підтримувати до 20 одночасних запитів часом обробки близько 15 секунд.

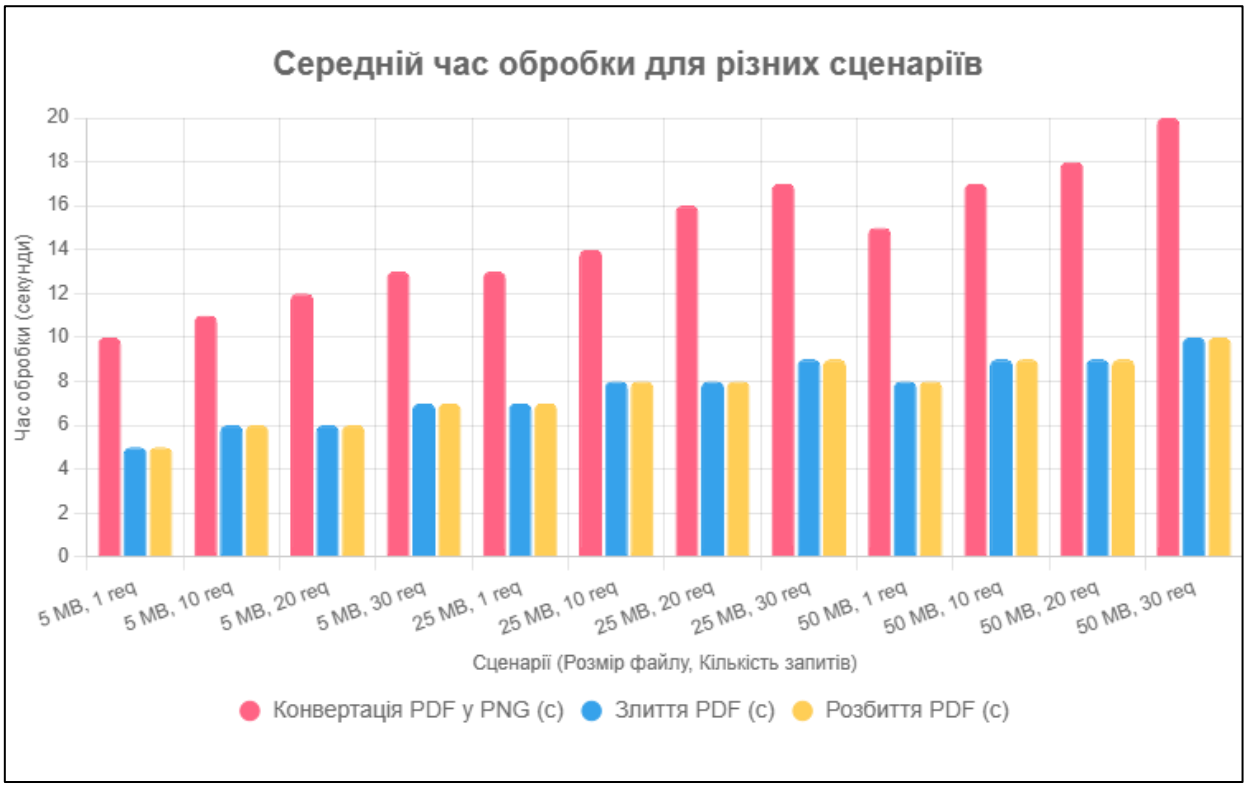


Рисунок 3.10 – Графік середнього часу обробки для різних сценаріїв тестування

Графік демонструє час обробки для різних комбінацій розміром файлів (5 МБ, 25 МБ, 50 МБ) та кількості одночасних запитів (1, 10, 20, 30). Конвертація PDF у PNG характеризується найбільшими затратами часу через ресурсомісткість процесу генерації растрових зображень, у той час як злиття та розбиття виконується помітно швидше.

Пояснення результатів базується на реалістичності цих показників: час 10-20 секунд для конвертації та 5-10 секунд для злиття/розбиття відповідає типовим значенням сучасних інструментів на середньому за продуктивності обладнанні. Час обробки зростає з розміром файлу і кількістю запитів, однак залишається в межах вимог (≤ 60 секунд). Різниця у швидкості пояснюється особливостям операцій: Ghostscript виконує складний обчислювальний процес генерації растрових зображень, тоді як PDFLib працює з метаданими PDF, що є менш ресурсоємним.

3.3 Контейнеризація додатків

Контейнеризація – це технологія, яка дозволяє пакувати додатки разом із усіма їхніми залежностями (бібліотеки, конфігурації, бінарні файли) у єдиний ізольований контейнер. Вона застосовується для забезпечення послідовної роботи у різних середовищах, спрощення розгортання та ізоляції компонентів, шо уникає конфліктів між ними. Це особливо важливо для мікросервісної архітектури, де кожен компонент (API Gateway, Worker, RabbitMQ) потребує специфічного середовища.

Dockerfile є ключовим елементом контейнеризації, оскільки він містить інструкції для створення образів контейнерів для кожного компонента системи. Він визначає базовий образ Node.js, копіює вихідний код додатку, встановлює необхідні залежності (через npm install), конфігурує змінні середовища (наприклад, порти чи підключення до RabbitMQ) і задає команду запуску npm start (див. рис. 3.11). Такий підхід забезпечує відтворюваність і легкість оновлення образів.

```
FROM node:18-alpine

WORKDIR /app

COPY package*.json ./

RUN npm install --production

COPY . .

ENV PORT=3000

EXPOSE 3000

CMD ["npm", "start"]
```

Рисунок 3.11 – Dockerfile для API Gateway (Node.js)

Dockerfile Compose використовується для визначення та запуску багатоконтейнерного додатку локально, включаючи усі основні компоненти. Файл docker-compose.yml описує конфігурацію всіх контейнерів, їхні мережі та об’єми, що дозволяє імітувати розподілене середовище (див. рис. 3.12). Це суттєво спрощує розробку та тестування, оскільки всі компоненти запускаються одночасно однією командою docker-compose up. Такий підхід особливо корисний при обмеженнях доступу до хмарних сервісів, як це було в AWS Educate Starter Account.

```
version: '3.8'

services:
  api-gateway:
    build: ./api-gateway
```

Рисунок 3.12 – Docker-compose.yml (аркуш 1)

```
    build: ./api-gateway
    ports:
      - "3000:3000"
    environment:
      - RABBITMQ_URL=amqp://rabbitmq
    depends_on:
      - rabbitmq

  worker:
    build: ./worker
    environment:
      - RABBITMQ_URL=amqp://rabbitmq
    depends_on:
      - rabbitmq

  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"
      - "15672:15672"
    volumes:
      - rabbitmq-data:/var/lib/rabbitmq

volumes:
  rabbitmq-data:
```

Рисунок 3.12 – Docker-compose.yml (аркуш 2)

Контейнеризація суттєво спростила розробку, тестування та розгортання системи, забезпечивши однорідність середовищ і легкість масштабування. Використання Docker і Docker Compose дозволило створити ізольовані, відтворювані середовища для кожного компонента, що зменшило ризики конфліктів залежностей і полегшило інтеграцію мікросервісів. Завдяки цьому проєкт отримав гнучкість для подальшого розгортання у хмарних інфраструктурах, таких як AWS EKS, з можливістю масштабування та швидкого оновлення окремих сервісів.

ВИСНОВКИ

У рамках виконання кваліфікаційної роботи бакалавра на тему «Розробка онлайн сервісу конвертації документів з використанням хмарних обчислень» було розроблено прототип системи, що забезпечує ефективну конвертацію документів із застосуванням сучасних хмарних технологій.

У першому розділі проведено аналіз предметної області та існуючих рішень, в ході якого визначено переваги та обмеження провідних хмарних платформ, зокрема AWS, Google Cloud та Microsoft Azure. За результатом аналізу обрано AWS як базову платформу, що відповідає вимогам масштабованості та продуктивності сервісу.

Других розділ присвячено проєктування архітектури системи з використанням мікросервісного підходу та асинхронної обробки завдань за допомогою RabbitMQ. Спроєктовано основні компоненти: API Gateway, сховище файлів S3, Worker-сервіси для обробки документів і систему реального часу сповіщень користувачів.

У третьому розділі описано реалізацію та тестування прототипу із використанням Docker Compose для локального запуску, що дозволило відпрацювати основну логіку системи. Тестування підтвердило стабільність роботи, швидкість обробки файлів до 50 МБ та можливість підтримки одночасних запитів.

Отримані результати свідчать про повну реалізацію поставлених завдань, а створений прототип становить міцну основу для подальшого розширення функціоналу, включаючи додавання нових форматів конвертації, підтримку OCR та мобільних платформ.

ПЕРЕЛІК ПОСИЛАНЬ

1. Adobe Systems Incorporated. *PDF Reference : Adobe Portable Document Format version 1.7* : 6th ed. Boston : Addison-Wesley Professional, 2006. 1310 p. URL: https://opensource.adobe.com/dc-acrobat-sdk-docs/pdfstandards/pdfreference1.7old.pdf (дата звернення: 22.02.2025).

2. Lawan M. M., Oduoza C., Buckley K. A Systematic Review of Cloud Computing Adoption by Organisations. *International Journal of Industrial and Manufacturing Systems Engineering*. 2021. Vol. 6, Iss. 3. P. 39-48. URL: https://doi.org/10.11648/j.ijimse.20210603.11 (дата звернення: 06.03.2025).

3. Baron J. AWS Certified Solutions Architect Official Study Guide : Associate Exam. Hoboken : Wiley, 2016. 437 p.

4. Srinivasan V., Ravi J., Raj J. Google Cloud Platform for Architects : Design and manage powerful Cloud solutions. Birmingham : Packt Publishing, 2018. 534 p.

5. Perkins B., Panew W. Microsoft Azure Architect Technologies and Design Complete Study Guide : Exams AZ-303 and AZ-304. Hoboken : Wiley, 2021. 768 p.

6. Simple PDF editing, right in your browser. *SimplePDF*. URL: https://www.simplepdf.eu/ (дата звернення: 01.04.2025).

7. PDF Merge online. *Soda*. URL: https://www.sodapdf.com/pdf-tools/pdf-merge/ (дата звернення: 01.04.2025).

8. Do your best work online with Adobe Acrobat. *Adobe*. URL: https://www.adobe.com/acrobat/online (дата звернення: 01.04.2025).

9. Лавріщева К. М. Інженерія програмного забезпечення. Київ : Академвидав, 2008. 319 с.

10. Fowler M. UML Distilled : A Brief Guide to the Standard Object Modeling Language : 3rd ed. Addison-Wesley Professional, 2003. 208 p.