**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ**
**Кафедра програмної інженерії**

# КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА СЕРВІСУ КОНВЕРТАЦІЇ PDF ДОКУМЕНТІВ»**

Виконав: студент 2 курсу, групи 8.1214-з
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
І.В. Колесніков
(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. Матвіїшина Н.В.
(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 магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

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

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2025 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Колеснікову Іллі Валерійовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка сервісу конвертації PDF документів

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 » жовтня 2025 року № 1457-с

2. Строк подання студентом роботи 28.11.2025 р.

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі, аналіз предметної області.
2. Проєктування.
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 22.10.2025 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. | 23.10.2025 | |
| | | | |
| 2. | Збір вихідних даних. | 24.10.2025 | |
| | | | |
| 3. | Обробка методичних та теоретичних | 27.10.2025 | |
| | джерел. | | |
| | | | |
| 4. | Розробка першого та другого розділу. | 07.11.2025 | |
| | | | |
| 5. | Розробка третього розділу. | 24.11.2025 | |
| | | | |
| 6. | Оформлення та нормоконтроль | 27.11.2025 | |
| | кваліфікаційної роботи магістра. | | |
| | | | |
| 7. | Захист кваліфікаційної роботи. | 12.12.2025 | |

Студент _____ І.В. Колесніков
(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 магістра «Розробка сервісу конвертації PDF документів»: 56 с., 32 рис., 1 табл., 21 джерело, 5 додатків.

ANGULAR, DOCKER, EXPRESS, LIBREOFFICE, MICROSERVICES, NODE.JS, PDF, REST API, RXJS, TRAEFIK, UML.

Об’єкт дослідження – вебсервіси обробки PDF-документів.

Мета роботи – розробити легкий мікросервісний вебсервіс для об’єднання, вирізання сторінок і конвертації DOCX у PDF.

Методи дослідження – аналітичний, моделювання, проєктування, програмний.

У роботі розглянуто проєктування та реалізацію сервісу для базових операцій над PDF. Клієнтська частина виконана засобами Angular та Material. Серверна частина містить два незалежні мікросервіси на Express: pdf-service, який відповідає за об’єднання/виділення сторінок на основі pdf-lib і convert-service, що забезпечує конвертацію DOCX у PDF на основі LibreOffice. Маршрутизацію та балансування навантаження між репліками забезпечує Traefik. Розгортання виконано в контейнерах Docker із можливістю горизонтального масштабування, використовуючи workers. Взаємодія між клієнтом і сервером реалізована через REST API.

Побудовано UML-діаграми: варіантів використання, діяльності, послідовності, класів та розгортання. Інтерфейс має просту структуру з великими зонами завантаження, підказками та картками інструментів на головній сторінці. Передбачено обмеження розміру файлів, валідацію типів і обробку помилок.

Таким чином, розроблений сервіс забезпечує простий і зрозумілий інтерфейс для кінцевого користувача, швидко обробляє документи, легко масштабується та підтримується завдяки контейнеризації й мікросервісній архітектурі.

# SUMMARY

Master’s qualifying paper “Development of a PDF Document Conversion Service”: 56 pages, 32 figures, 1 table, 21 references, 5 supplements.

ANGULAR, DOCKER, EXPRESS, LIBREOFFICE, MICROSERVICES, NODE.JS, PDF, REST API, RXJS, TRAEFIK, UML.

The object of the study is web services for processing PDF documents.

The aim of the study is to develop a lightweight microservice web service for merging, cutting pages, and converting DOCX to PDF.

The methods of research are analytical, modeling, design, programming.

The paper considers the design and implementation of a service for basic operations on PDF. The client part is implemented using Angular and Material. The server part contains two independent microservices on Express: pdf-service, which is responsible for merging/extracting pages based on pdf-lib, and convert-service, which provides DOCX to PDF conversion based on LibreOffice. Traefik provides routing and load balancing between replicas. Deployment is performed in Docker containers with the possibility of horizontal scaling using workers. Interaction between the client and server is implemented via REST API.

UML diagrams were built: use cases, activities, sequences, classes, and deployment. The interface has a simple structure with large download areas, tooltips, and tool cards on the main page. File size limits, type validation, and error handling are provided.

Thus, the developed service provides a simple and intuitive interface for the end user, quickly processes documents, is easily scalable, and is supported by containerization and microservice architecture.

**ЗМІСТ**

Завдання на кваліфікаційну роботу ..... 2
Реферат ..... 4
Summary ..... 5
Вступ ..... 8
1 Технічне завдання ..... 10
1.1 Терміни та визначення ..... 10
1.1.1 Загальні терміни ..... 10
1.1.2 Технічні терміни ..... 10
1.2 Функціональні вимоги ..... 11
1.2.1 Призначення і цілі створення системи ..... 11
1.2.2 Загальні функціональні можливості системи ..... 11
1.3 Нефункціональні вимоги ..... 12
1.4 Опис предметної області ..... 13
1.5 Опис системи ..... 13
1.6 Огляд та порівняння аналогів ..... 14
2 Проєктування ..... 16
2.1 Використання UML під час розробки системи ..... 16
2.2 Діаграма варіантів використання ..... 17
2.2.1 Опис варіантів використання ..... 19
2.3 Діаграма діяльності ..... 21
2.4 Діаграма класів ..... 23
2.5 Діаграма послідовності ..... 25
2.6 Діаграма розгортання ..... 26
3 Програмна реалізація та тестування ..... 28
3.1 Опис інструментів розробки ..... 28
3.2 Основні класи та модулі системи ..... 29
3.3 Встановлення та налаштування оточення ..... 30

3.4 Опис API ..... 34
3.5 Тестування проєкту ..... 36
3.6 Керівництво користувача ..... 37
3.6.1 Домашня сторінка ..... 37
3.6.2 Об’єднання PDF (Merge) ..... 38
3.6.3 Виділення діапазону (Slice) ..... 40
3.6.4 Конвертація DOCX у PDF (DOCX to PDF) ..... 42
Висновки ..... 44
Перелік посилань ..... 45
Додаток А Angular MergeComponent ..... 47
Додаток Б Angular ApiService ..... 49
Додаток В Express pdf.Js ..... 50
Додаток Г Express index.Js ..... 51
Додаток Д Unit-test MergeComponent ..... 53

# ВСТУП

З впровадженням та швидким переходом робочих та навчальних процесів на електроний документообіг зростає попит на інструменти, що дозволяють швидко обробляти документи без інсталяції важких програм. Онлайн-сервіси для роботи з PDF відіграють важливу роль, адже забезпечують доступність інструментів з будь-якого пристрою.

Ринок пропонує чимало рішень для роботи з PDF, однак вони або мають надмірний функціонал і складні у користуванні, або є комерційними/закритими, або не дозволяють легко масштабувати обробку під навантаження. Враховуючи зростаючі потреби, постає задача створити легкий, адаптивний, зрозумілий та масштабований сервіс із мінімальним, але достатнім набором функцій.

Актуальність роботи полягає у створенні мікросервісного вебсервісу, який поєднує простоту інтерфейсу з найкращими інженерними практиками: контейнеризація, балансування й маршрутизація, можливість запуску декількох workers та горизонтального масштабування.

З огляду на це, сформульовано цілі та задачі дослідження.

Мета: розробити легкий мікросервісний вебсервіс для об’єднання, вирізання сторінок і конвертації DOCX у PDF.

Завдання:

– сформулювати функціональні та нефункціональні вимоги до сервісу;
– проаналізувати, описати та порівняти дотичні рішення;
– спроєктувати архітектуру сервісу;
– реалізувати клієнтську частину та серверні мікросервіси;
– підготувати контейнеризацію та оркестрацію;
– протестувати ключові сценарії та надати керівництво користувача.

Об’єкт дослідження: вебсервіси обробки PDF-документів.

Предмет дослідження: Angular, Express.js, Docker/Traefik та інтеграція з LibreOffice.

Методи дослідження: аналітичний, моделювання, проєктування, програмний.

У першому розділі описані функціональні/нефункціональні вимоги, подано аналіз предметної області та огляд аналогів із таблицею порівняння.

У другому розділі обґрунтовано використання UML, спроєктовано архітектуру й наведено основні діаграми (варіантів використання, діяльності, послідовності, класів, розгортання).

У третьому розділі описано інструменти розробки, встановлення та налаштування середовища, подано опис API, приклади unit-тестування та керівництво користувача.

# 1 ТЕХНІЧНЕ ЗАВДАННЯ

## 1.1 Терміни та визначення

### 1.1.1 Загальні терміни

Система – сервіс для базових операцій з PDF-документами.

Angular – платформа та фреймворк для розробки SPA-застосунків [2].

Angular Material – бібліотека UI-компонентів для Angular, що забезпечує сучасний, адаптивний інтерфейс [3].

Express.js – мінімалістичний серверний фреймворк для Node.js [7].

ДВВ – діаграма варіантів використання.

ДД – діаграма діяльності.

ДК – діаграма класів.

ДП – діаграма послідовності.

ДР – діаграма розгортання.

Користувач – суб’єкт, що експлуатує систему.

### 1.1.2 Технічні терміни

PDF – формат електронних документів, що зберігає розмітку сторінок.

DOCX – формат документів Microsoft Word.

UI – інтерфейс користувача.

SPA-застосунок – вебзастосунок, який завантажується на одній сторінці і динамічно оновлює контент без перезавантаження всього сайту.

Мікросервіс – автономний сервіс, що реалізує вузьку частину бізнес-логіки і незалежно масштабується.

API Gateway – фасадний сервіс, що приймає запити від клієнта та маршрутизує їх до відповідних мікросервісів.

Worker – робочий процес/екземпляр мікросервісу, що виконує обчислювальні завдання.

Об’єктне сховище – сховище файлів, де тимчасово розміщуються вхідні/вихідні документи.

Компонент – структурна одиниця Angular.

## 1.2 Функціональні вимоги

### 1.2.1 Призначення і цілі створення системи

Функціональне призначення системи – забезпечити швидке виконання базових операцій над PDF-документами, а саме: об’єднання кількох PDF, розділення за сторінками/діапазонами, конвертація DOCX у PDF.

Експлуатаційне призначення – система експлуатується користувачами для виконання базових операцій над PDF-документами.

Мета створення системи – розробити мінімалістичний, зручний і масштабований сервіс PDF-операцій з сучасним UI на Angular та хмарною мікросервісною архітектурою на Express.

### 1.2.2 Загальні функціональні можливості системи

Система має надавати користувачам наступні можливості:

– перегляд головної сторінки;
– використання навігації для переходу між сторінками;
– об’єднувати PDF;
– розділяти PDF;
– конвертувати DOCX у PDF;
– налаштувати параметри операцій;

– завантажувати DOCX/PDF-файли;
– отримувати PDF-файли.

### 1.3 Нефункціональні вимоги

**Інтерфейс користувача**. Коректне відображення на екранах за стратегією mobile first. Шапка (header) повинна відображатися на всіх сторінках та відображати: меню навігації, лого з переходом на головну. Підвал (footer) повинен відображатися на всіх сторінках та містити список використаних інстументів. Головна сторінка повинна містити 3 плитки відповідних операцій: об’єднання, розділення, конвертація. Сторінки функцій повинні містити зону для завантаження файлів, панель налаштувань (де доречно), функціональні кнопки, індикатор прогресу

**Підтримка браузерів**. Система повинна коректно працювати в актуальних версіях Safari, Firefox, Chrome, Edge.

**Вимоги до продуктивності**. Початкове завантаження <= 1 с. на сучасних пристроях за умови стабільного Інтернет-з’єднання. Операції повинні виконуватися протягом кількох секунд на сервері за умови доступних ресурсів. Навантаження на мікросервіс повинно паралелізуватися за рахунок кількох worker-ів.

**Вимоги до безпеки та приватності**. Доступ до серверних операцій повинен здійснюватися виключно через API Gateway з валідацією вхідних даних та CORS-політикою. Повинно діяти обмеження на розмір (25 МБ/файл) і типи (PDF, DOCX) файлів. При обробці файлів повинні відбуватися наступні операції: санітизація імен файлів, перевірка MIME-типів, фільтрація метаданих. Файли зберігаються тимчасово у приватному контейнері. Для захисту від перевантаження повинні використовуватися ліміти запитів, захист від повторної відправки, обмеження кількості паралельних завдань.

## 1.4 Опис предметної області

Предметною областю є виконання операцій з документами у форматі PDF та взаємопов’язані перетворення з DOCX.

Розглянемо типові сценарії.

Об’єднання – користувач відкриває сторінку “Merge”, завантажує 2 або більше PDF-файлів, змінює порядок перетягуванням, запускає обробку, після чого отримує один злитий PDF.

Розділення – користувач відкриває сторінку “Slice”, завантажує PDF, задає сторінки та/або діапазони, запускає обробку, отримує PDF лише з потрібними сторінками.

Конвертація DOCX у PDF – користувач відкриває сторінку “DOCX to PDF”, завантажує DOCX, запускає обробку, отримує PDF із збереженням структури та базового форматування.

Права доступу не вимагають реєстрації: користувач виконує усі операції через UI.

## 1.5 Опис системи

Система побудована на мікросервісній архітектурі з базовим набором сервісів:

- UI – SPA-застосунок, реалізований на Angular + Material, з трьома окремими сторінками під кожну операцію, header/меню, лого з переходом на головну, footer із переліком інструментів;
- API Gateway – реалізована на Express єдина точка входу для клієнта, яка перевіряє запити та маршрутизує їх до мікросервісів;
- мікросервіси:
  - pdf-service – об’єднання/розділення сторінок з PDF;
  - convert-service – конвертація DOCX у PDF;

– сховище файлів – тимчасове приватне об’єктне сховище для вхідних/вихідних результатів;
– масштабування – використовуються декілька worker-ів на сервіс (горизонтальне масштабування відбувається через оркестрацію/кластеризацію).

Наданий поділ надає можливість незалежно масштабувати потенційні ресурсозатратні операції (наприклад, конвертація DOCX у PDF), забезпечуючи стабільну роботу під навантаженням.

### 1.6 Огляд та порівняння аналогів

На ринку існують численні онлайн-сервіси роботи з PDF. Розглянемо деякі з таких сервісів.

Вебсервіс Smallpdf має широкий вибір інструментів, серед яких: об’єднання, розділення, конвертації між форматами, стиснення, редагування. Із переваг можна виділити покриття великої кількості сценаріїв, але це ж призводить до ускладнення взаємодії з інтерфейсом. Сервіс має ліміти безкоштовного використання функцій, потребує додаткових кроків взаємодії та має рекламні елементи, що збільшує шлях до результату у базових операціях [18].

Сервіс iLovePDF орієнтований на масове повсякденне використання, надає швидкий доступ до поширених задач. Як і Smallpdf, надає великий каталог функцій, тому користувач витрачає додатковий час на пошук потрібної. Також має обмеження функціоналу в безкоштовному плані [9].

Sejda PDF виділяється зручним редагуванням. Підходить, коли потрібно виконувати змішані сценарії (наприклад, редагування з базовими перетвореннями). Натомість, багатофункціональність ускладнює маршрут для базових операцій, а політики продуктивності/лімітів у безкоштовному плані можуть впливати на зручність під навантаженням [17].

Adobe Acrobat Online є частиною екосистеми Adobe з акцентом на якість конвертацій і сумісність. Має потужні можливості, але часто потребує входу в обліковий запис, інтегрується з ширшою лінійкою продуктів, що додає кроків і контексту, не завжди необхідних для швидких точкових операцій [1].

У таблиці 1.1 наведено порівняння аналогів з нашою системою за базовими критеріями.

Таблиця 1.1 – Порівняння аналогів

| **Функція / Платформа** | **Наша система** | **Smallpdf** | **iLovePDF** | **Sejda PDF** | **Adobe Acrobat Online** |
|---|---|---|---|---|---|
| Об’єднання PDF | Так | Так | Так | Так | Так |
| Розділення | Так | Так | Так | Так | Так |
| Конвертація DOCX у PDF | Так | Так | Так | Так | Так |
| Фокус на мінімалізмі UI | Так | Ні | Ні | Частково | Ні |
| Мікросервісна архітектура | Так | – | – | – | – |

Аналіз систем демонструє, що попри те, що всі розглянуті платформи підтримують базові операції з PDF, вони орієнтовані на велику кількість інструментів, що збільшує когнітивне навантаження і кількість кроків для типових задач. Наша система орієнтована на цільовий мінімалістичний сервіс із трьома ключовими діями: об’єднання, розділення, конвертація.

Використання мікросервісної архітектури дозволяє легше масштабувати точково і простіше контролювати політики приватності. Такий підхід зменшує час до результату, спрощує підтримку, а також надає гнучкі можливості до розширення.

# 2 ПРОЄКТУВАННЯ

## 2.1 Використання UML під час розробки системи

UML – стандартизована мова моделювання, що використовується для візуалізації, специфікації, конструювання та документування програмних систем [20].

UML застосовується незалежно від методології та технологічного стека, вона допомагає узгодити вимоги та архітектуру до початку імплементації, що особливо корисно для клієнт-серверних і мікросервісних систем, де важливі чіткі межі відповідальності, інтерфейси та взаємодії [8, 20].

Підтримується консорціумом Object Management Group (OMG). UML виступає мовою проєктування, а не програмування, даючи змогу формалізувати логіку та структуру системи до початку реалізації [20].

Розглянемо переваги UML для нашої системи [20].

Стандартизованість – офіційна специфікація OMG забезпечує єдині правила і позначення, що зменшує неоднозначності в артефактах аналізу/проєктування.

Візуалізація – діаграми полегшують спільну роботу замовника, аналітика та команди над архітектурою й вимогами.

Підтримка ООП – нотація добре поєднується з інкапсуляцією, наслідуванням і поліморфізмом, що корисно для моделювання доменних сутностей (документи, задачі обробки, статуси).

Зменшення витрат – рання формалізація сценаріїв і структур зменшує помилки під час імплементації мікросервісів і API.

У цій роботі застосовано наступні діаграми:

– діаграма варіантів використання – для функціональних вимог і акторів;
– діаграма діяльності – для алгоритмів і гілок процесу;
– діаграма класів – для статичної структури сутностей;

– діаграма послідовності – для часових взаємодій компонентів;

– діаграма розгортання – для фізичного розміщення вузлів/артефактів.

Таке покриття відповідає класифікації UML на структурні, поведінкові та діаграми взаємодії. Разом ці діаграми узгоджують логіку, структуру даних і механізми взаємодії між інтерфейсом і мікросервісами.

## 2.2 Діаграма варіантів використання

Діаграма варіантів використання відображає функціональні вимоги і взаємодію акторів із системою через прецеденти. Вона фокусується на зовнішньо спостережуваних результатах для акторів, а не на внутрішній реалізації [21].

Розглянемо основні компоненти ДВВ [21].

Актори визначаються як зовнішні об’єкти, що взаємодіють із системою, виконуючи певні дії. Актори можуть бути класифіковані як користувачі, інші системи або пристрої.

Сам варіант використання використовується для опису функціональних можливостей системи з точки зору акторів, які використовують цю систему.

Відношення визначається як лінія, що вказує на взаємодію актора з конкретним варіантом використання. Відношення може бути прямим або спеціалізованим (include/extend).

ДВВ пропонує чіткий підхід до документування системних вимог, що позитивно впливає на доступність для всіх учасників приєкту, понижуючи технічний порог [8, 10].

Діаграма варіантів використання легко перетворюється на інші типи діаграм, що полегшує перехід від процесу аналізу вимог до проєктування та впровадження [21].

На рисунку 2.1 зображена діаграма варіантів використання для сервісу конвертації PDF.

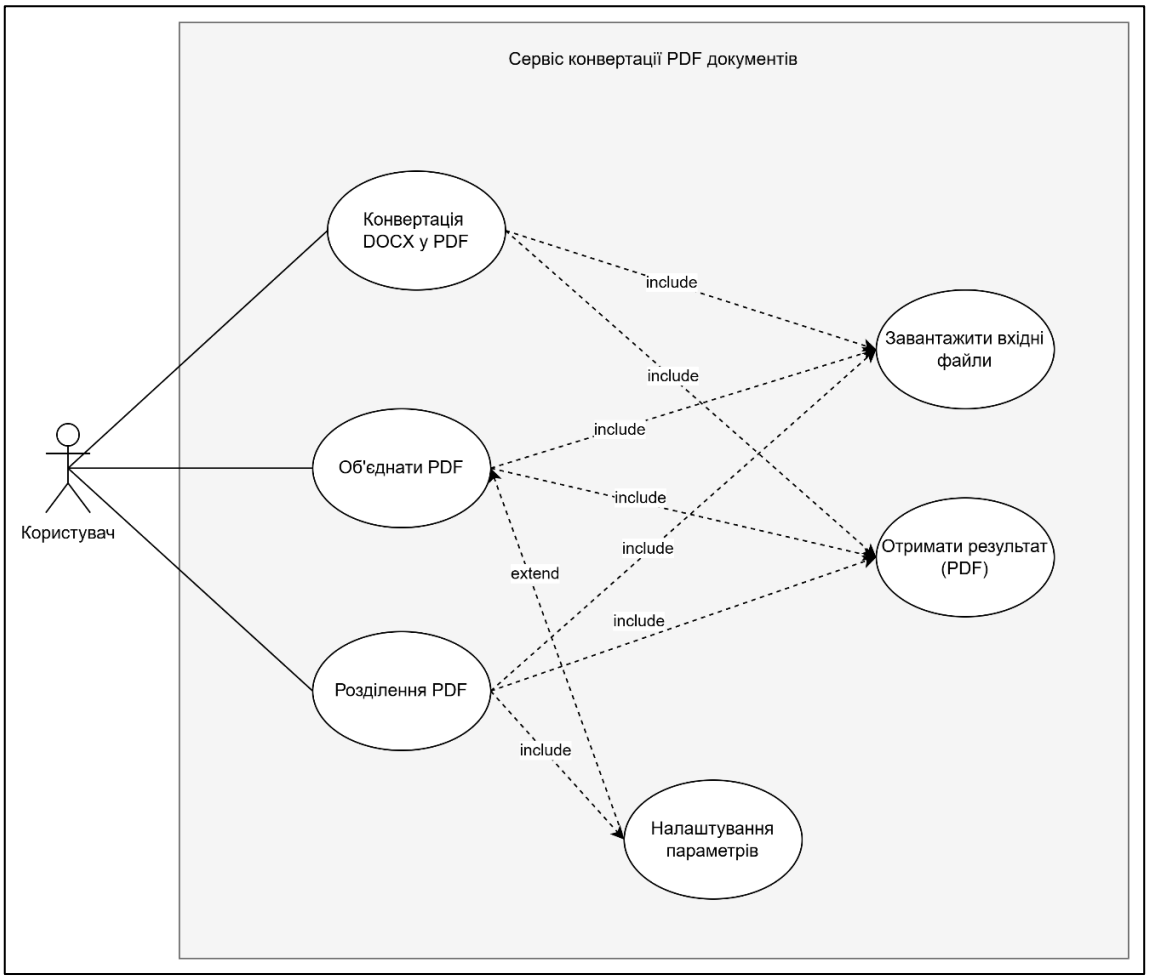



Рисунок 2.1 – Діаграма варіантів використання

Представлено актора «Користувач», який взаємодіє з системою на рівні UI.

Виділимо один із основних прецедентів системи – «Об’єднати PDF». Цей варіант використання є ключовим і доступний усім користувачам. Після натискання на плитку “Merge PDF” або відповідний пункт меню, відображається сторінка операції з дроп-зоною та панеллю налаштувань.

Користувач завантажує щонайменше два PDF-файли, за потреби змінює їх порядок перетягуванням і натискає «Об’єднати».

Система виконує валідацію (типи/розміри), надсилає запит через API Gateway до мікросервісу pdf-merge, очікує результат і надає посилання/автозавантаження злитого файлу.

У разі помилок (некоректний формат, перевищення ліміту, збій сервера) користувач отримує повідомлення та рекомендацію щодо виправлення або повтору спроби.

### 2.2.1 Опис варіантів використання

**Прецедент «Об’єднати PDF».**

Призначення: даний варіант використання надає можливість користувачу об’єднати декілька PDF у один файл.

Передумови: користувач знаходиться на сторінці “Merge”.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає «Вибрати файли» або перетягує PDF у зону завантаження. Система валідовує тип і розмір файлів, відображає список, користувач за потреби змінює порядок і натискає кнопку «Об’єднати». Система виконує обробку та надає посилання для завантаження результату.

Виняткова ситуація 1: невалідні дані (менше двох файлів, хибний тип/MIME або перевищено розмір) – система виводить повідомлення про помилку та блокує кнопку «Об’єднати».

Виняткова ситуація 2: помилка обробки – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

**Прецедент «Розділення PDF».**

Призначення: даний варіант використання надає можливість користувачу сформувати новий PDF за заданими діапазонами сторінок.

Передумови: користувач знаходиться на сторінці “Slice”.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач завантажує PDF, вводить діапазони сторінок (наприклад, «1-3,5,10-12») та натискає кнопку «Розділити». Система валідовує формат діапазонів, виконує вирізання сторінок і надає посилання для завантаження результату.

Виняткова ситуація 1: невалідні дані (порожні або хибні діапазони, вихід за межі документа) – система підсвічує поля з помилкою та блокує кнопку «Розділити».

Виняткова ситуація 2: помилка обробки – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

**Прецедент «Конвертація DOCX у PDF».**

Призначення: даний варіант використання надає можливість користувачу конвертувати документ DOCX у PDF.

Передумови: користувач знаходиться на сторінці “DOCX to PDF”.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач завантажує DOCX-файл і натискає кнопку «Конвертувати». Система валідовує тип і розмір файла, виконує конвертацію та надає посилання для завантаження результату.

Виняткова ситуація 1: невалідні дані (невірний тип/MIME або перевищено розмір) – система виводить повідомлення про помилку та блокує кнопку «Конвертувати».

Виняткова ситуація 2: помилка обробки – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

**Прецедент «Завантажити вхідні файли».**

Призначення: даний варіант використання надає можливість користувачу завантажити потрібні файли для обраної операції.

Передумови: користувач знаходиться на сторінці відповідної операції (об’єднання, розділення або конвертація).

Основний потік подій: даний варіант використання починає виконуватися, коли користувач натискає «Вибрати файли» або перетягує їх у зону завантаження. Система перевіряє типи й розміри, відображає список/прев’ю та фіксує готовність до подальшої обробки.

Виняткова ситуація 1: невалідні дані (хибний тип/MIME або перевищено розмір) – система виводить повідомлення про помилку та скасовує завантаження конкретних файлів.

Виняткова ситуація 2: збій передавання/зберігання – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

**Прецедент «Отримати результат (PDF)».**

Призначення: даний варіант використання надає можливість користувачу завантажити сформований PDF-файл після обробки.

Передумови: користувач успішно виконав відповідну операцію, результат доступний для завантаження.

Основний потік подій: даний варіант використання починає виконуватися, коли система успішно завершує одну з операцій. Файл передається на клієнт, система відображає повідомлення про успішне завантаження.

Виняткова ситуація 1: строк доступності результату минув – система виводить повідомлення про недоступність файлу та пропонує виконати операцію повторно.

Виняткова ситуація 2: збій мережі/помилка з’єднання – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

**Прецедент «Налаштування параметрів».**

Призначення: даний варіант використання надає можливість користувачу встановити параметри операції.

Передумови: користувач завантажив вхідні файли на сторінці відповідної операції.

Основний потік подій: даний варіант використання починає виконуватися, коли користувач вводить або коригує параметри у відповідних полях/контролах. Система валідовує формат значень, підсвічує помилки та зберігає коректні параметри для подальшої обробки.

Виняткова ситуація 1: невалідні параметри – система виводить повідомлення про помилку та блокує запуск операції.

Виняткова ситуація 2: внутрішня помилка збереження параметрів – система виводить повідомлення про технічну проблему та пропонує повторити спробу.

## 2.3 Діаграма діяльності

Діаграма діяльності моделює потік робіт, а саме: дії, переходи, гілкування/об’єднання, а також синхронізацію паралельних гілок. Для сервісу

конвертації PDF діаграма надає можливість формалізувати алгоритми “Merge”, “Slice” і “DOCX to PDF”: від завантаження файлів і валідації (тип/розмір/MIME) до постановки завдання на обробку worker-ом і повернення результату користувачу. Це допомагає виявляти альтернативні шляхи (помилки валідації, перевищення лімітів, тайм-аути) та точки можливого паралелізму [8, 10].

На рисунках 2.2 – 2.3 надано ДД прецеденту «Об’єднати PDF».

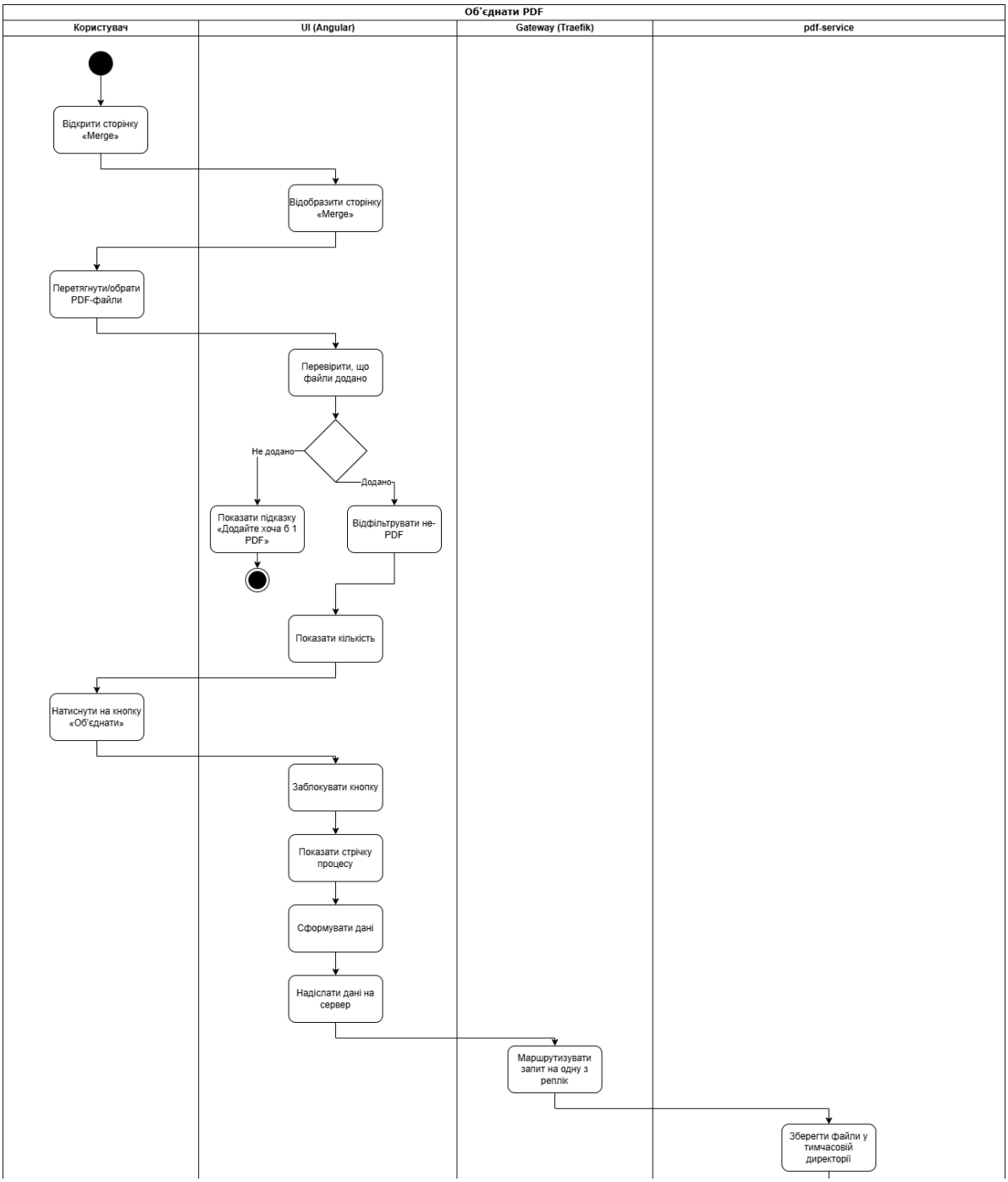



Рисунок 2.2 – ДД прецеденту «Об’єднати PDF»

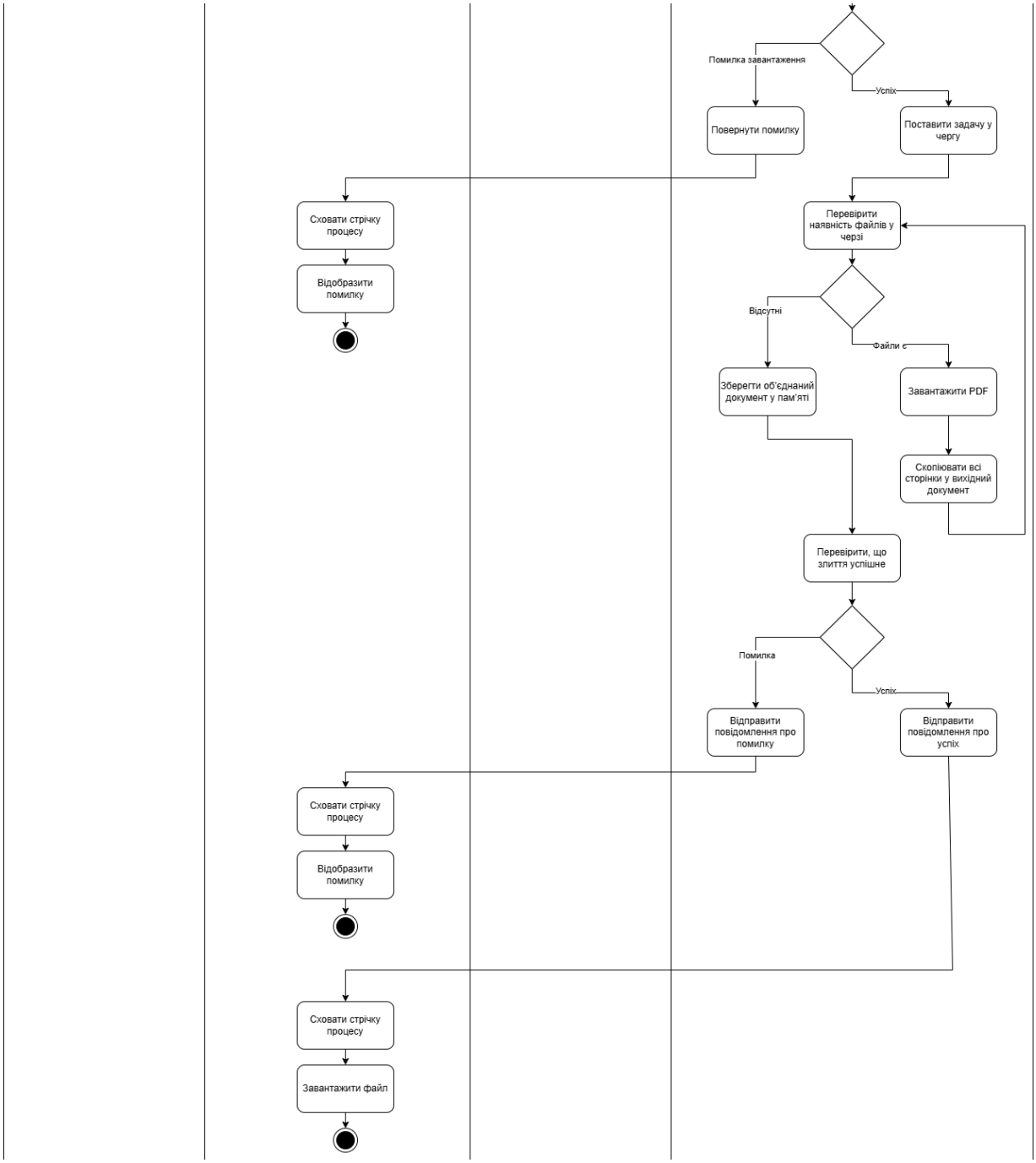



Рисунок 2.3 – ДД прецеденту «Об’єднати PDF»

## 2.4 Діаграма класів

ДК відображає статичну структуру системи, а саме: класи, їх атрибути та операції, відносини (асоціації, агрегація/композиція, узагальнення), кратності й видимість. У контексті сервісу конвертації PDF ця діаграма слугує основою доменної моделі й API-контрактів [8, 10].

На рисунках 2.4 – 2.5 зображено діаграми класів зі сторони UI та мікросервіса convert-service прецеденту «Конвертація DOCX у PDF».

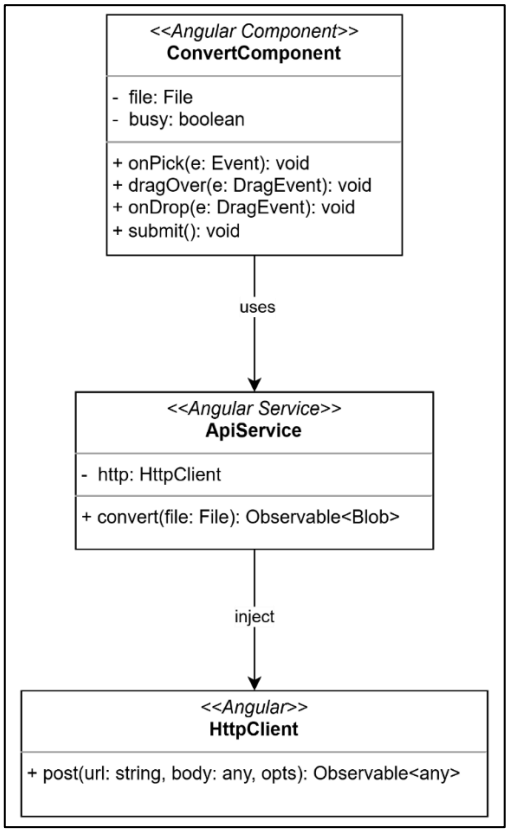



Рисунок 2.4 – ДК зі сторни UI прецеденту

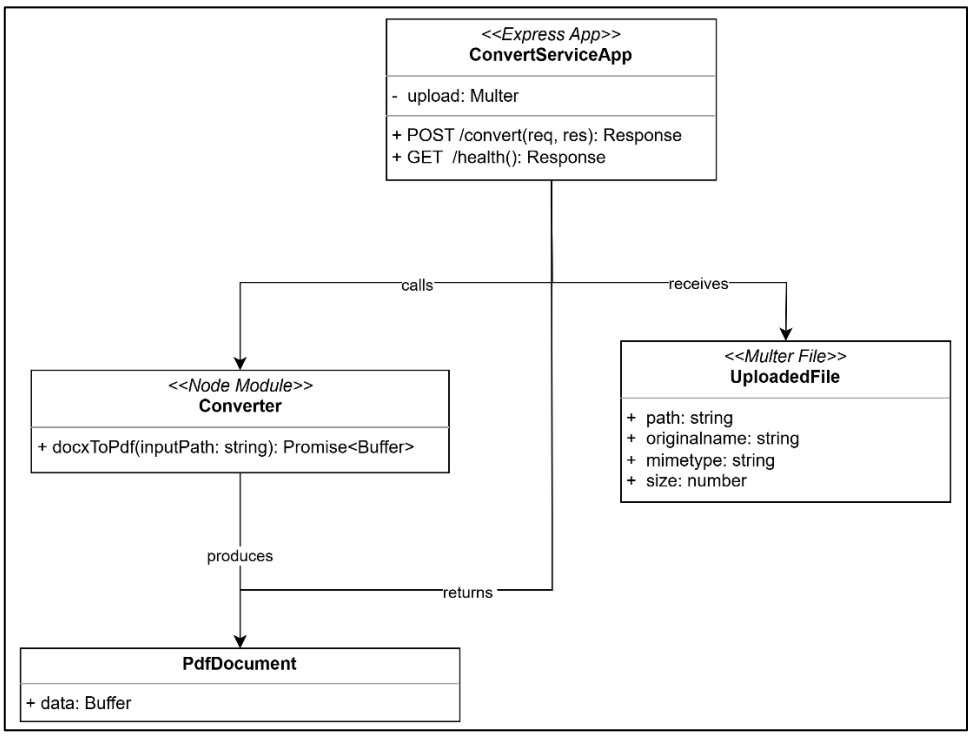



Рисунок 2.5 – ДК зі сторни convert-service

Модель класів узгоджується з типами DTO, що використовуються у REST-інтерфейсі, і спрощує генерацію/валідацію схем даних.

## 2.5 Діаграма послідовності

ДП відтворює взаємодії у часі між об’єктами/компонентами через повідомлення/виклики. У випадку нашого сервісу це ланцюжок «Angular UI -> API Gateway -> відповідний мікросервіс -> об’єктне сховище -> UI». Такий артефакт корисний для проєктування REST-взаємодій, логування та управління помилками, а також для перевірки, що прогрес і помилки коректно відображаються на клієнті [8, 10].

На рисунку 2.6 наведено ДП для сценарію «Конвертація DOCX у PDF».

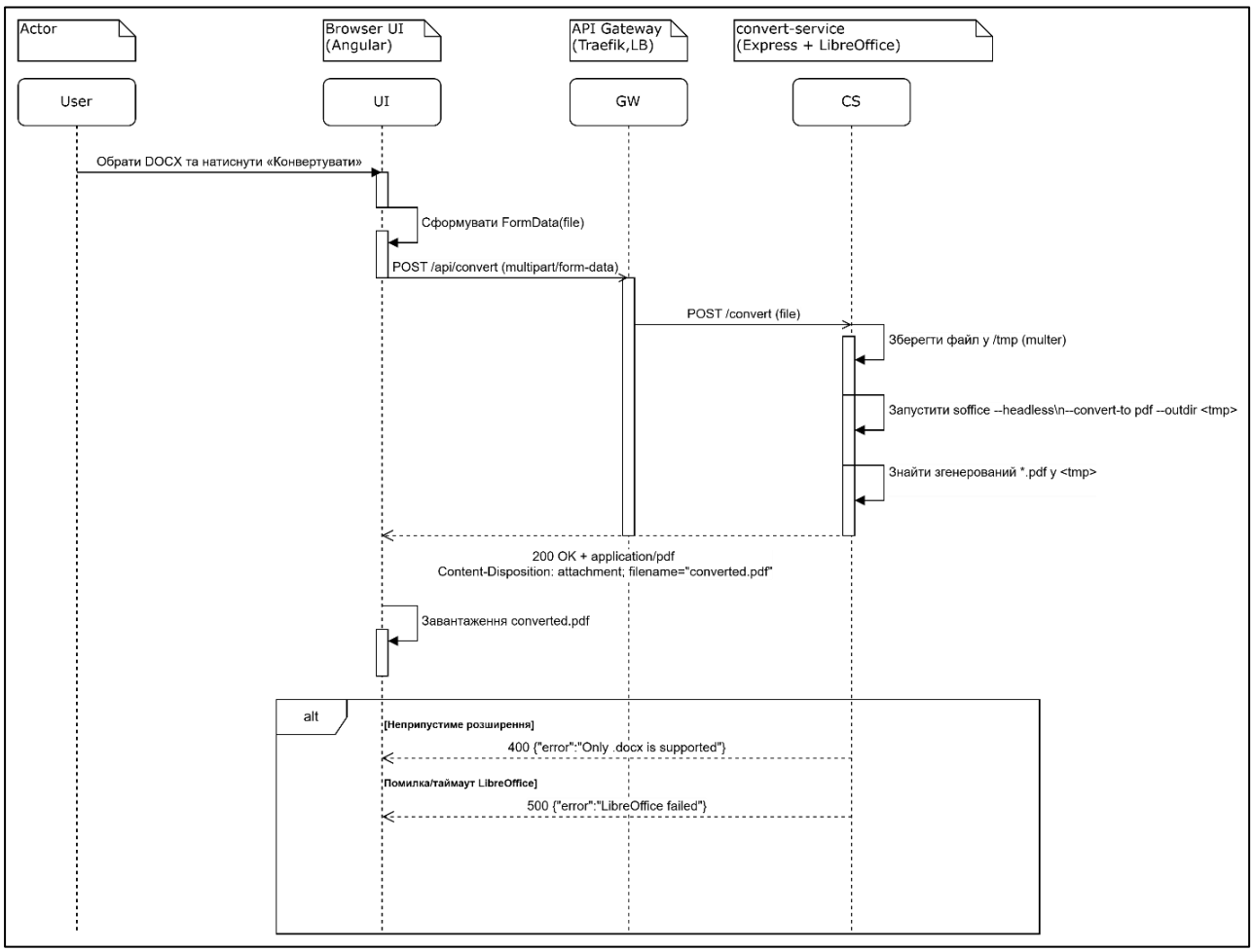



Рисунок 2.6 – ДП прецеденту «Конвертація DOCX у PDF»

## 2.6 Діаграма розгортання

ДР показує вузли виконання, артефакти, що на них розміщені, і мережеві з’єднання між ними. Це дає змогу формально описати виконувальну архітектуру, а саме: де саме розміщено SPA, як трафік проходить через API Gateway до мікросервісів і куди зберігаються вхідні/вихідні файли. У UML таке моделювання використовується для фіксації призначення артефактів на цільові вузли та параметрів виконання [8, 10].

На рисунку 2.7 показано розгортання системи.

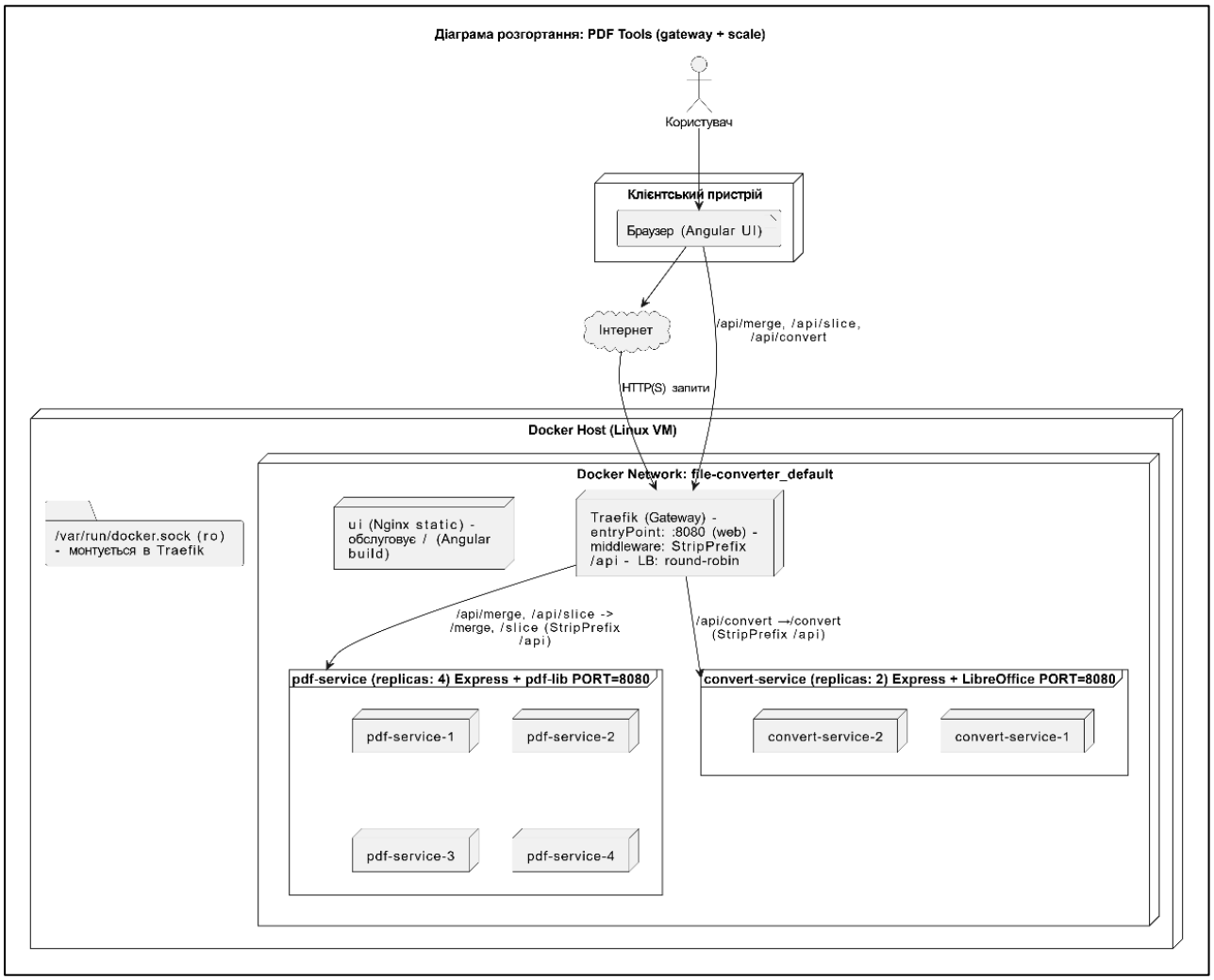



Рисунок 2.7 – Діаграма розгортання

Застосування UML-нотації дозволяє послідовно описати функціонал, алгоритм, інтеграційні взаємодії, розгортання і статичну структуру домену для

нашого сервісу. Це забезпечує узгодженість вимог, архітектури та реалізації, зменшує ризики на етапі імплементації й полегшує масштабування мікросервісів у майбутньому.

# 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

## 3.1 Опис інструментів розробки

Система реалізована за допомогою сучасного стеку, що забезпечує масштабованість, гнучкість та високу продуктивність при мінімальному функціоналі.

Angular – фреймворк для розробки SPA з standalone-компонентами, декларативним підходом і реактивністю [2].

Angular Material – бібліотека UI-компонентів для швидкого створення адаптивного інтерфейсу згідно з Material Design [3].

RxJS – реактивні потоки, підписки та оператори для роботи з асинхронними подіями [5, 16].

Node.js – серверна платформа з неблокуючою моделлю вводу/виводу [11].

Express.js – фреймворк для маршрутизації та middleware [7].

pdf-lib – бібліотека для роботи з PDF на стороні pdf-service [14].

Multer – прийом multipart/form-data та збереження тимчасових файлів у /tmp [13].

LibreOffice – конвертація DOCX у PDF у convert-service.

Traefik – API-gateway та балансувальник навантаження між worker-ами сервісів [19].

Docker / Docker Compose – контейнеризація, стандартизація середовищ, opchestration на рівні Compose [6].

p-queue – керування внутрішньою чергою завдань/паралельністю в pdf-service.

Jasmine/Karma – unit-тестування клієнтської частини [2, 4].

### 3.2 Основні класи та модулі системи

Основним бізнес-процесом є робота з файлами PDF. Відповідно, основними класами/модулями є ті, що забезпечують цей процес.

Зі сторони клієнту можна виділити наступні класи.

– HomeComponent – Angular-компонент, що є контейнером для інших компонентів, відображає стартову сторінку з трьома адаптивними картками-інструментами;
– MergeComponent – Angular-компонент, що відповідає за інтерфейс для завантаження кількох PDF і запуску об’єднання;
– SliceComponent – Angular-компонент, що відповідає за інтерфейс для завантаження PDF та задання діапазону сторінок;
– ConvertComponent – Angular-компонент, що відповідає за інтерфейс для завантаження DOCX і запуску конвертації в PDF;
– ApiService – Angular- сервіс, що відповідає за взаємодію з сервером.

Зі сторони серверу можна виділити наступні складові:

– pdf-service – мікросервіс, що відповідає за об’єднання/розділення PDF;
– convert-service – мікросервіс, що відповідає за конвертацію DOCX у PDF.

Нижче наведено фрагмент функції об’єднання PDF на сервері (рис. 3.1).

```
import { PDFDocument } from 'pdf-lib';

export async function mergePdf(buffers) :Promise<any> { Show usages
  const out = await PDFDocument.create();
  for (const buf of buffers) {
    const src = await PDFDocument.load(buf);
    const pages = await out.copyPages(src, src.getPageIndices());
    pages.forEach(p => out.addPage(p));
  }
  return out.save();
}
```

Рисунок 3.1 – Функція об’єднання PDF

З повними реалізаціями MergeComponent/ApiService можна ознайомитися у додатках А, Б, серверні обробники – у додатках В, Г.

### 3.3 Встановлення та налаштування оточення

Для початку розробки системи необхідно налаштувати відповідне оточення. Оскільки всі частини нашої системи працюють на JavaScript, то необхідно встановити актуальні версії Node та npm. Також необхідно встановити Docker та Docker Compose.

У якості операційної системи обрано Linux Ubuntu, тому у термінал необхідно ввести команди sudo apt install nodejs та sudo apt install npm для встановлення Node та npm [11, 12], для Docker та Docker Compose виконуємо відповідні команди з документації [6].

Після встановлення створюємо структуру нашого проєкта. Для початку створимо батьківську теку file-converter з дочірньої текою services. Далі створимо клієнтську частину на Angular, для цього необхідно встановити Angular CLI. Робимо це за допомогою команди npm install -g @angular/cli [2], після за допомогою команди ng new ui –standalone –style=scss –routing із кореню проєкту створюємо клієнтську частину [2, 4].

У теці services створюємо заготовки тек мікросервісів pdf-service і convert-service з файлами-заглушками для Dockerfile та файлів, які будуть відповідати за внутрішню логіку. У корені проєкту створюємо заглушку для docker-compose.yml. Отримуємо прооєкт з чіткою та зрозумілою структурою (рис. 3.2).

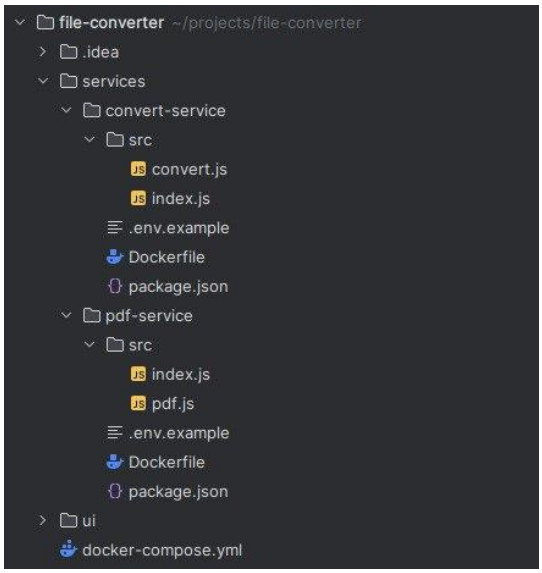



Рисунок 3.2 – Структура проєкту

Далі переходимо до Dockerfile відповідних сервісів, почнемо з pdf-service. Для нього використовуємо Node 20-alpine, runtime-залежності встановлюємо через npm install --omit=dev, назначаємо порт – 8080 (рис. 3.3) [6, 11].

```
FROM node:20-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install --omit=dev
COPY src ./src
ENV PORT=8080
EXPOSE 8080
CMD ["node", "src/index.js"]
```

Рисунок 3.3 – Dockerfile pdf-service

За аналогією заповнюємо Dockerfile для convert-service. Але цу ньому вже використовуємо node:20-slim та додаємо в образ пакети libreoffice-core, libreoffice-writer, fonts-dejavu, інші інструкції залишаються без змін (рис. 3.4) [6].

```
FROM node:20-slim
ENV DEBIAN_FRONTEND=noninteractive
RUN apt-get update \
 && apt-get install -y --no-install-recommends \
    libreoffice-core libreoffice-writer libreoffice-calc \
    fonts-dejavu ghostscript ca-certificates \
 && apt-get clean && rm -rf /var/lib/apt/lists/*

WORKDIR /app
COPY package*.json ./
RUN npm install --omit=dev
COPY src ./src
ENV PORT=8080
EXPOSE 8080
CMD ["node", "src/index.js"]
```

Рисунок 3.4 – Dockerfile convert-service

Далі переходимо до docker-compose.yml. Так як UI може посилати запити до різних мікросервісів, необхідно зробити єдину точку входу, яка буде вирішувати куди відправити запит та зможе балансувати навантаження між

кількома екземплярами сервісу. Для цього використовуємо Traefik, який є і реверс-проксі, і балансувальником навантаження [19].

Також необхідно підключити мікросервіси та налаштувати відповідні Traefik-інструкції (рис. 3.5) [6, 19].

```yaml
services:
  traefik:
    image: traefik:v3.0
    container_name: traefik
    command:
      - --providers.docker=true
      - --providers.docker.exposedbydefault=false
      - --entryPoints.web.address=:80
    ports:
      - "8080:80"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock:ro
    restart: unless-stopped

  pdf-service:
    build: ./services/pdf-service
    environment:
      - PORT=8080
      - MAX_FILE_SIZE_MB=25
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.pdf.rule=PathPrefix(`/api/merge`) || PathPrefix(`/api/slice`)"
      - "traefik.http.routers.pdf.entrypoints=web"
      - "traefik.http.routers.pdf.middlewares=api-strip@docker"
      - "traefik.http.middlewares.api-strip.stripprefix.prefixes=/api"
      - "traefik.http.services.pdf.loadbalancer.server.port=8080"
    restart: unless-stopped

  convert-service:
    build: ./services/convert-service
    environment:
      - PORT=8080
      - MAX_FILE_SIZE_MB=25
    labels:
      - "traefik.enable=true"
      - "traefik.http.routers.convert.rule=PathPrefix(`/api/convert`)"
      - "traefik.http.routers.convert.entrypoints=web"
      - "traefik.http.routers.convert.middlewares=api-strip@docker"
      - "traefik.http.middlewares.api-strip.stripprefix.prefixes=/api"
      - "traefik.http.services.convert.loadbalancer.server.port=8080"
    restart: unless-stopped
```

Рисунок 3.5 – Docker-compose.yml

Docker-compose.yml піднімає Traefik v3 як реверс-проксі та два бекенд-сервіси: pdf-service і convert-service. Traefik слухає HTTP-з’єднання, читає інструкції з контейнерів Docker і на їх основі віддає запити потрібному сервісу.

Сервіс traefik запускається з образу traefik:v3.0, приймає трафік на внутрішньому порту 80, який пробрасуємо назовні як 8080. Він працює як

Docker-провайдер, але з параметром exposedbydefault=false, тож жоден контейнер не публікується без інструкції traefik.enable=true. Для відмовостійкості вказуємо політику перезапуску unless-stopped [19].

Pdf-service збирається з локальної директорії ./services/pdf-service і всередині слухає порт 8080. Змінна MAX_FILE_SIZE_MB=25 відповідає за ліміт розміру файлів. Набір інструкцій повідомляє Traefik, що контейнер треба публікувати, і що всі запити з префіксами шляху /api/merge або /api/slice мають потрапляти саме до нього через entrypoint web. Перед прокиданням до сервісу застосовуємо middleware, який видаляє префікс /api, тому бекенд фактично отримує шляхи /merge або /slice. Traefik автоматично скеровує трафік у контейнер на порт 8080 [6, 19].

Convert-service організовано аналогічно, він теж слухає порт 8080 і має той самий ліміт файлів. Але правила маршрутизації наступні: будь-який запит з префіксом /api/convert спрямовується в цей контейнер. Middleware так само видаляє префікс /api, і бекенд отримує шлях /convert.

Масштабування контейнерів виконується Docker Compose параметром --scale. Його вказуємо при запуску команди docker compose up -d --scale pdf-service=4 --scale convert-service=2. Після запуску отримуємо 4 репліки pdf-service і 2 репліки convert-service – це горизонтальні worker-и, між якими Traefik розподіляє запити (рис. 3.6) [6, 19].

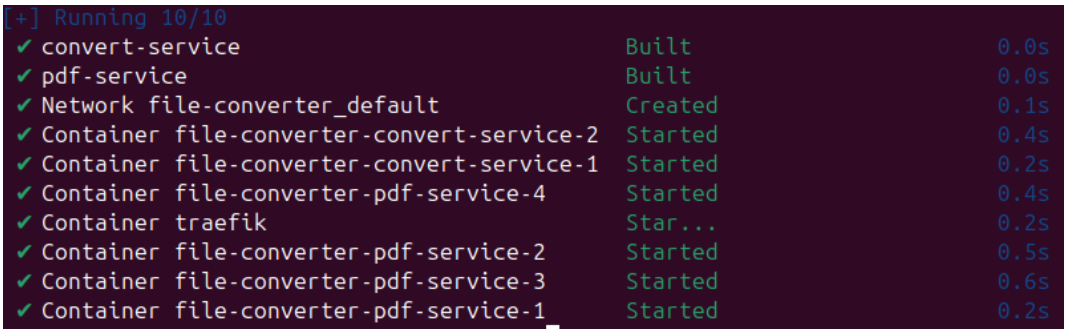



Рисунок 3.6 – Запуск контейнерів

Після запуску сервісів, переходимо до ui. У корені створимо файл proxy.conf.json, щоб можна було звертатися до gateway без CORS (рис. 3.7) [4].




Рисунок 3.7 – Файл proxy.conf.json

Для запуску ui використовуємо команду ng serve --proxy-config proxy.conf.json -o (рис. 3.8) [4].

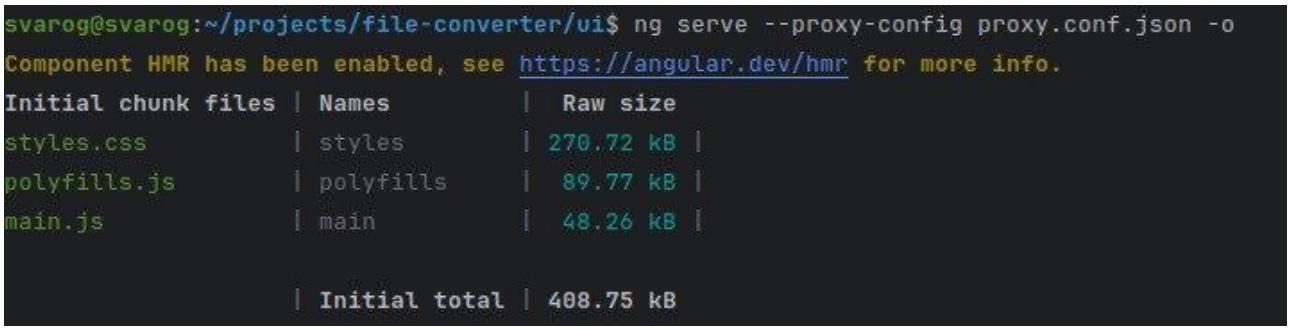



Рисунок 3.8 – Запуск ui

### 3.4 Опис API

У системі реалізовано REST API, яке забезпечує взаємодію між клієнтською та серверною частинами через HTTP-запити. Основні операції виконуються методом HTTP POST із використанням multipart/form-data.

Усі запити обробляються через фреймворк Express.js, а балансування та маршрутизацію забезпечує Traefik. Сервіси працюють у stateless-режимі, проміжні файли створюються у тимчасових каталогах та видаляються після завершення обробки.

Розглянемо основні API-endpoints.

POST /api/merge відповідає за об’єднання кількох PDF у один документ. Приймає multipart/form-data з полем files[]. Відповідь містить згенерований PDF з заголовком Content-Disposition: attachment; filename="merged.pdf".

POST /api/slice відповідає за виділення діапазону сторінок із PDF. Приймає multipart/form-data з полем file та числовими параметрами from, to.

Відповідь містить згенерований PDF з заголовком Content-Disposition: attachment; filename="slice.pdf".

POST /api/convert відповідає за конвертацію DOCX у PDF. Приймає multipart/form-data з полем file. Відповідь містить згенерований PDF з заголовком Content-Disposition: attachment; filename="converted.pdf".

Для помилок повертаються наступні коди та повідомлення [15]:

– 400 Bad Request – відсутні або некоректні поля/типи файлів;
– 413 Payload Too Large – перевищено ліміт розміру завантаження;
– 415 Unsupported Media Type – невідповідний MIME тип;
– 500 Internal Server Error – помилки під час обробки файлів.

Для опису структури API використовується Swagger UI, який надає інтерактивний інтерфейс для перегляду маршрутів, тестування запитів і перевірки валідації даних (рис. 3.9).

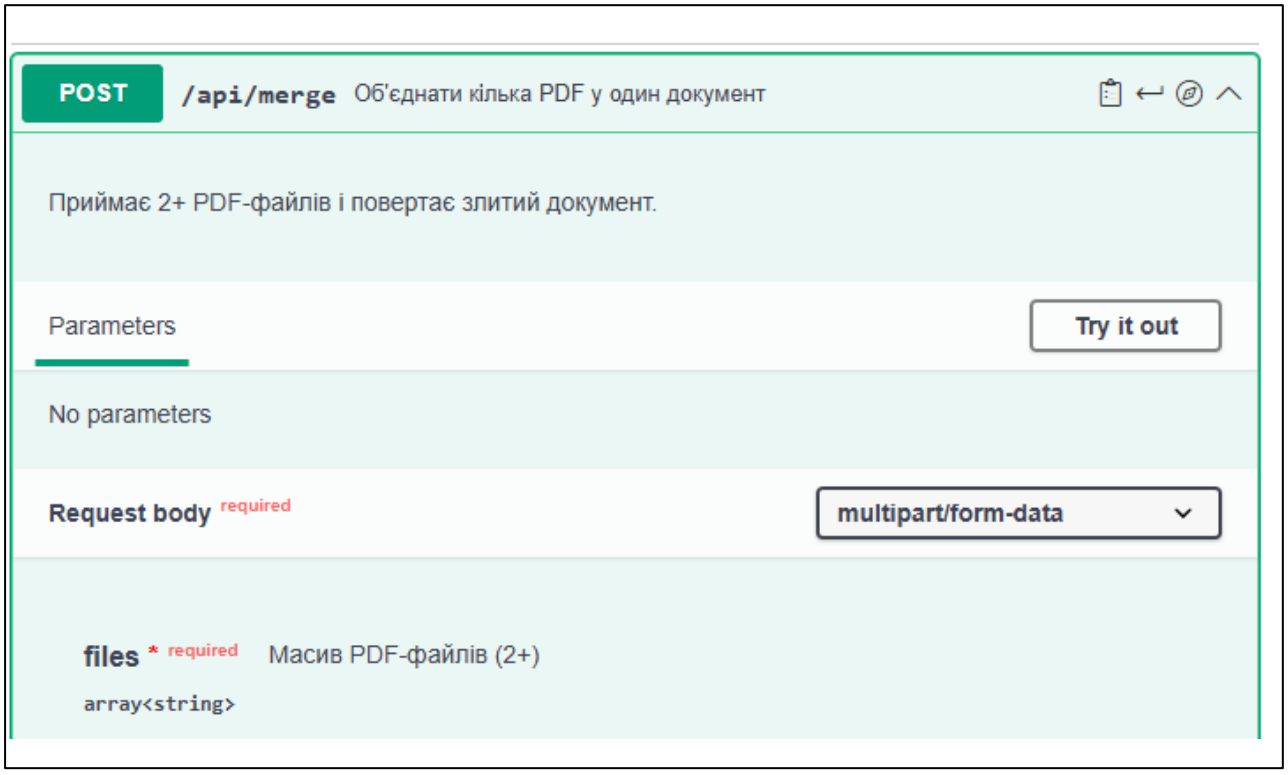



Рисунок 3.9 – Опис API у Swagger UI

Ознайомитися детально з описом API можна за посиланням https://app.swaggerhub.com/apis/znu-3ce/pdf-tools-api/1.0.0.

### 3.5 Тестування проєкту

У процесі розробки системи було проведено unit-тестування для забезпечення цілісності основної бізнес-логіки, надійності взаємодії з API та коректної роботи користувацького інтерфейсу.

Для тестування використовувалися інтегровані в Angular CLI фреймворки Jasmine та Karma [2]. Тестування мало дві головні цілі: по-перше, перевірити коректність роботи форм та завантаження файлів (drag&drop, відсів не-PDF/не-DOCX, стани кнопок і прогрес-індикаторів), а по-друге – перевірити взаємодію з ApiService (виклики /api/merge, /api/slice, /api/convert, обробка успішних/помилкових відповідей).

На рисунку 3.10 наведено приклад unit-тесту компонента MergeComponent, який перевіряє, що завантажені файли є PDF.

```
it('onDrop приймає лише PDF і очищує клас dragover', () => {
  const dz = fixture.debugElement.query(By.css('.dropzone'))
    .nativeElement as HTMLElement;

  const pdf :File  = pdfFile('good.pdf');
  const png = new File(['x'], 'img.png', { type: 'image/png' });

  const ev = new DragEvent('drop', { bubbles: true });
  Object.defineProperty(ev, 'dataTransfer', { value: { files: fileList(pdf, png) } });
  Object.defineProperty(ev, 'preventDefault', { value: () => {} });
  Object.defineProperty(ev, 'currentTarget', { value: dz });

  dz.classList.add('dragover');
  component.onDrop(ev);

  expect(component.files.length).toBe(1);
  expect(component.files[0].name).toBe('good.pdf');
  expect(dz.classList.contains('dragover')).toBeFalse();
});
```

Рисунок 3.10 – Unit-тест MergeComponent

Тести запускалися у браузері на шлюзі 9876 за допомогою консольної команди ng test або npm run test. Результати прогону виводяться як у вікні браузера, так і в консолі (рис. 3.11) [2, 4].

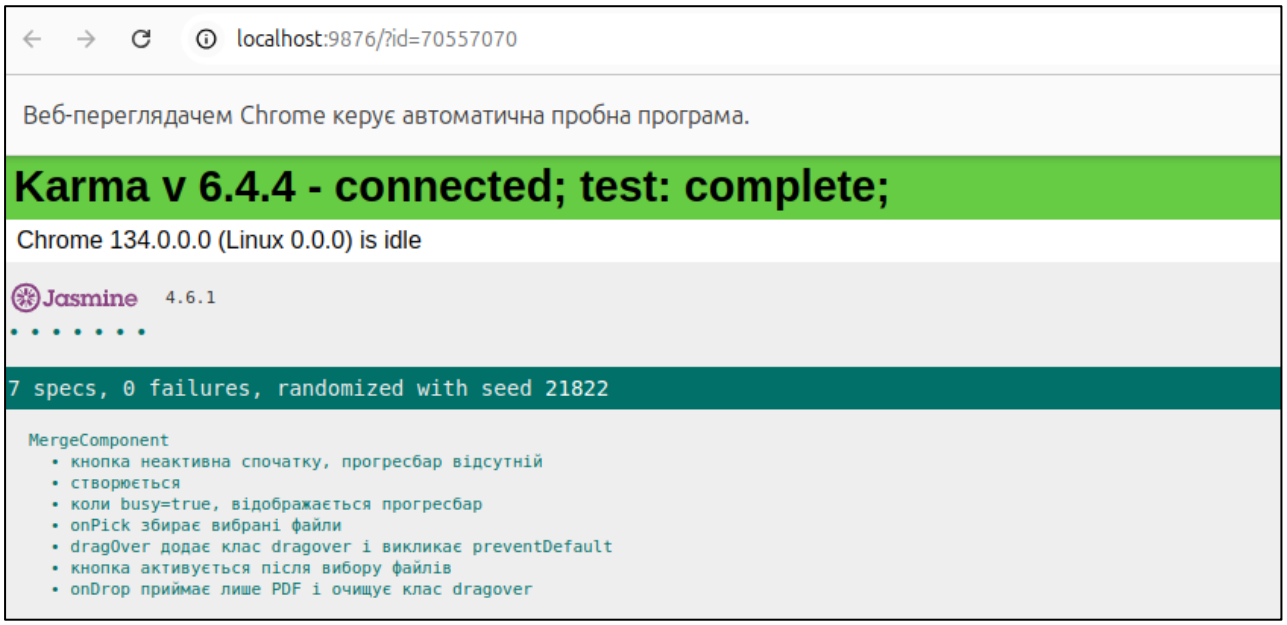



Рисунок 3.11 – Результат тестування MergeComponent

Проведене тестування дозволяє виявляти помилки в бізнес-логіці, забезпечує стабільну взаємодію з сервером та коректне відображення станів інтерфейсу [4].

Детально з прикладом тестів MergeComponent можна ознайомитися у додатку Д.

## 3.6 Керівництво користувача

### 3.6.1 Домашня сторінка

Після відкриття вебзастосунку відображається головна сторінка з трьома квадратними картками-інструментами: Merge PDF, Slice PDF, DOCX to PDF. Картки адаптивні, займають доступний простір і відкривають відповідну сторінку по кліку (рис. 3.12).

Для навігації можна використовувати як картки, так і верхнє меню (рис. 3.13).

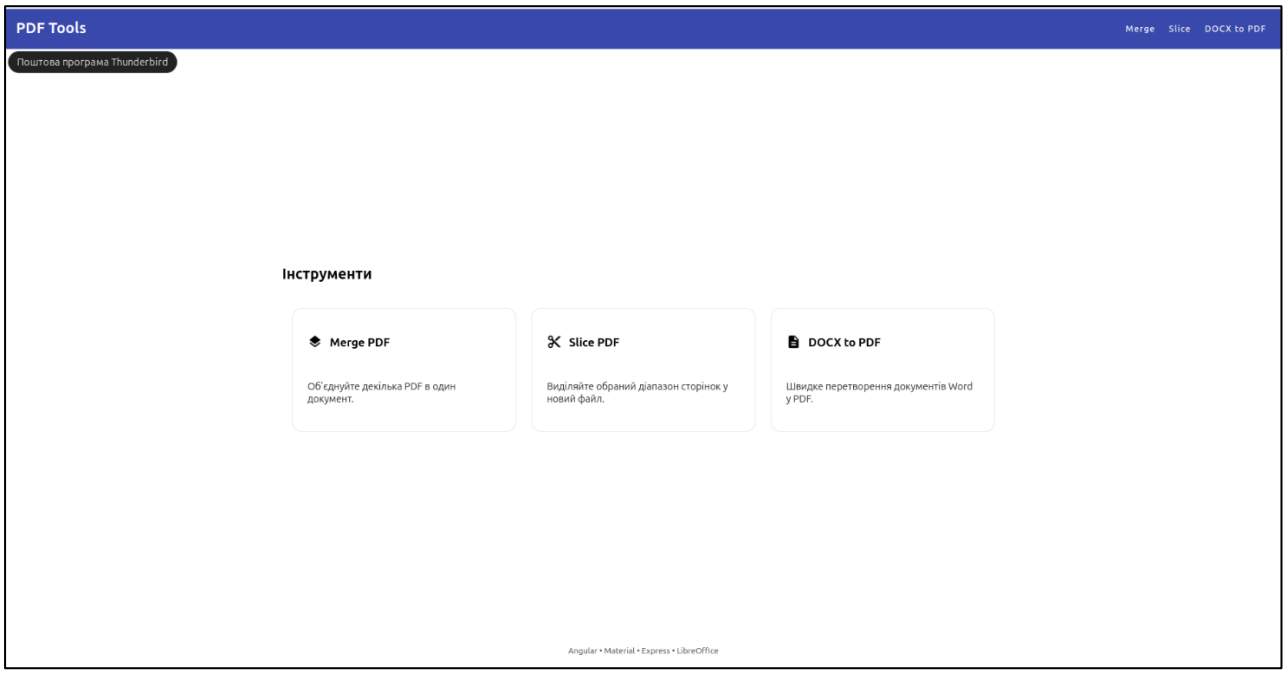



Рисунок 3.12 – Головна сторінка

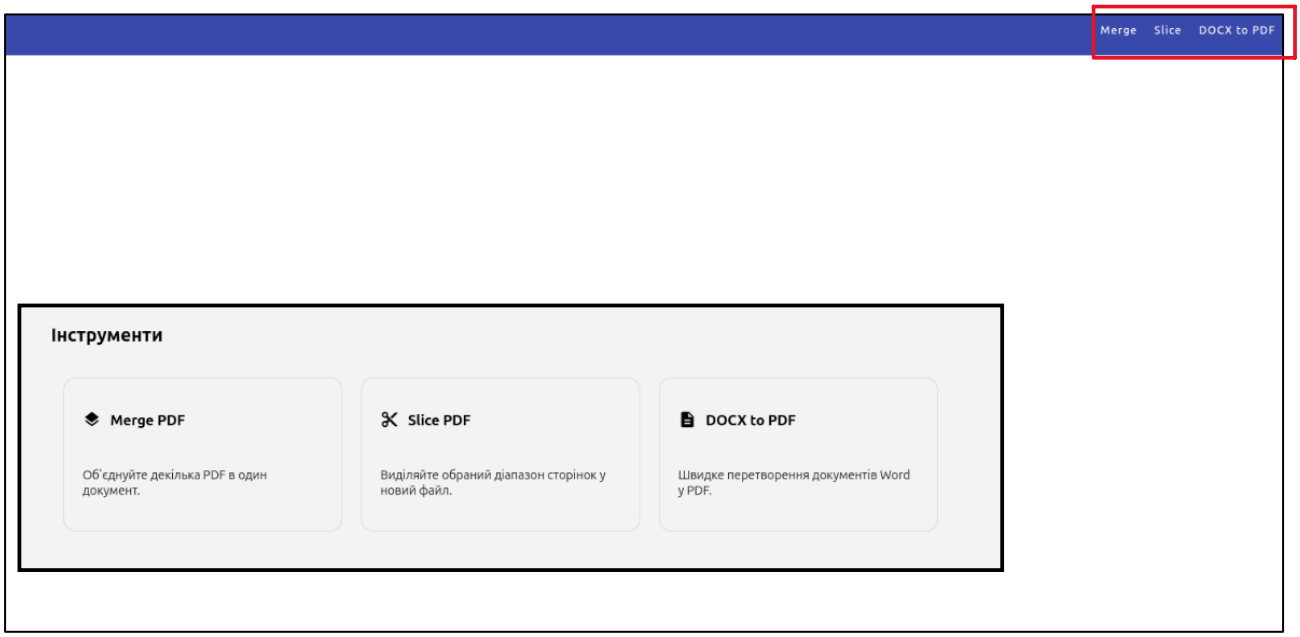



Рисунок 3.13 – Навігація

### 3.6.2 Об’єднання PDF (Merge)

Після того, як користувач переходить на сторінку Merge, система відображає відповідний інтерфейс. За замовчуванням кнопка «Об’єднати» заблокована, поки не буде додано щонайменше 2 PDF-файли (рис. 3.14).

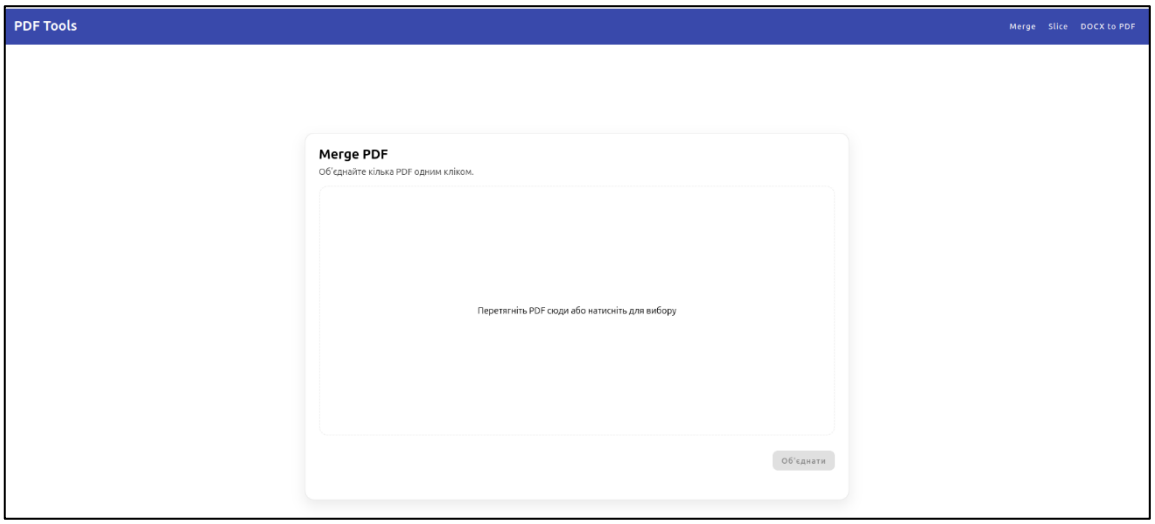



Рисунок 3.14 – Сторінка Merge

Для того щоб об’єднати декілька файлів, користувач повинен пере тягнути їх у drop-зону або обирати через файловий діалог, після натиснути на кнопку «Об’єднати». Система формує єдиний PDF і завантажує результат (рис. 3.15 – 3.16).

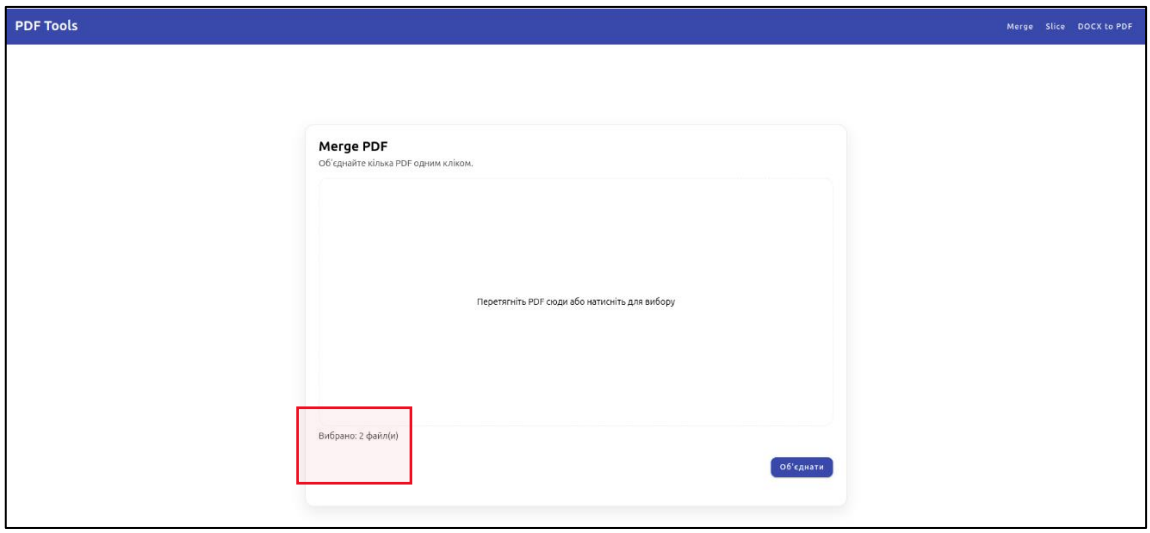



Рисунок 3.15 – Приєднані файли

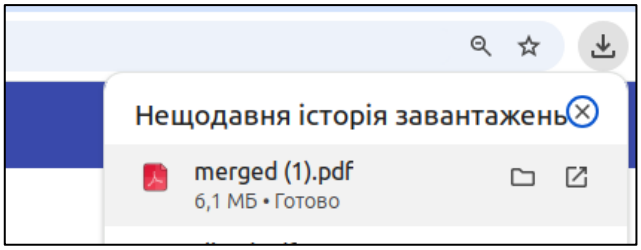



Рисунок 3.16 – Результат об’єднання

Під час процесу об’єдання система відображає користувачу стрічку очікування виконання процесу (рис. 3.17).

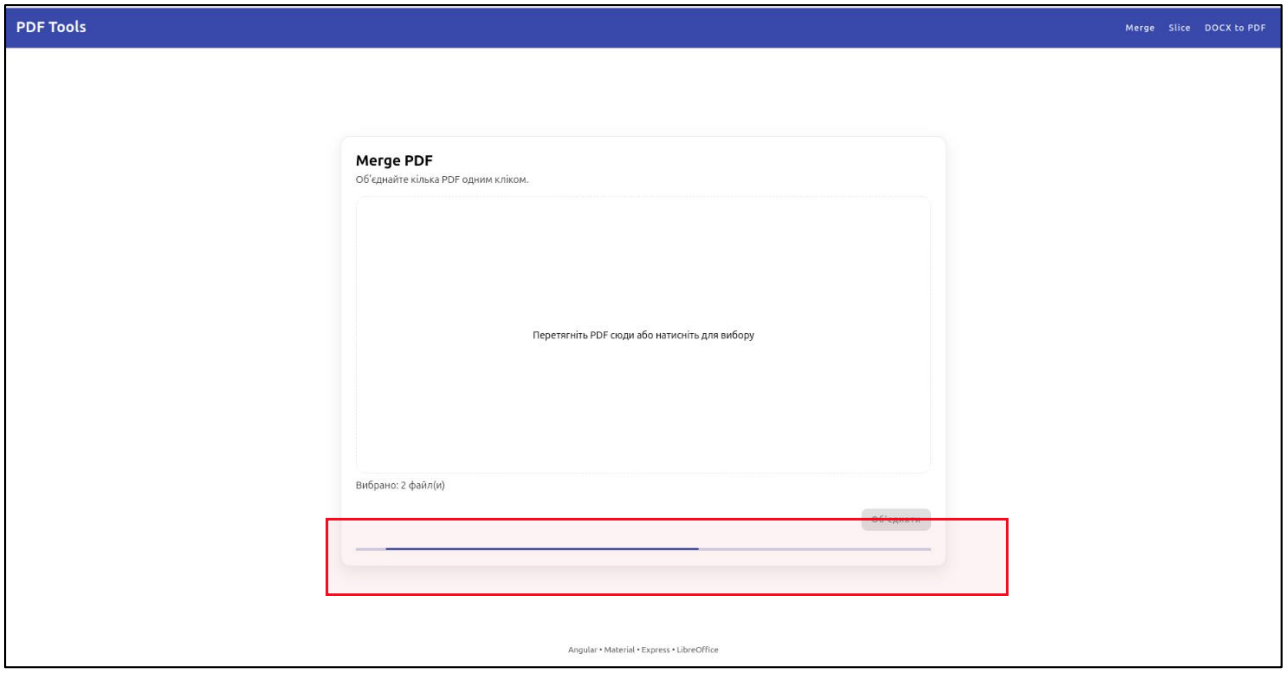



Рисунок 3.17 – Очікування виконання процесу

### 3.6.3 Виділення діапазону (Slice)

Після переходу на сторінку Slice, система відображає користувачу інтерфейс з двома колонками: зліва – drop-зона для файлу, а справа – параметри діапазону (рис. 3.18).

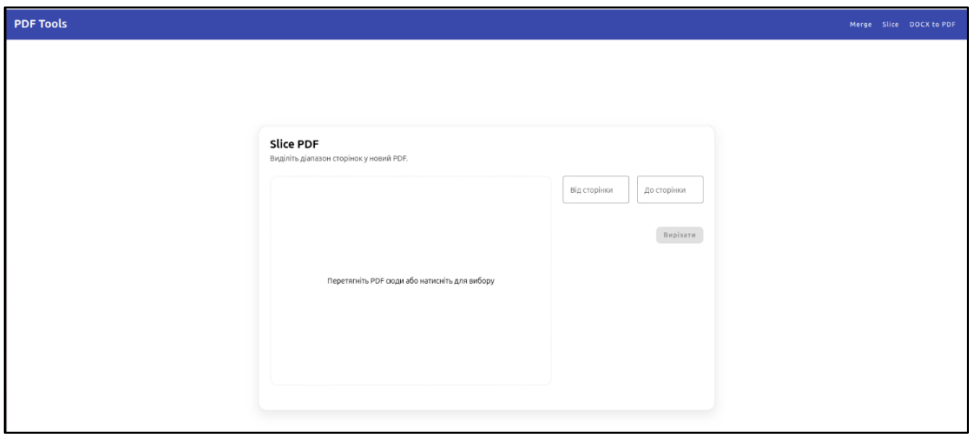



Рисунок 3.18 – Сторінка Slice

Після того, як користувач обрав файл, система відображає його назву і активує кнопку «Вирізати» (рис. 3.19).

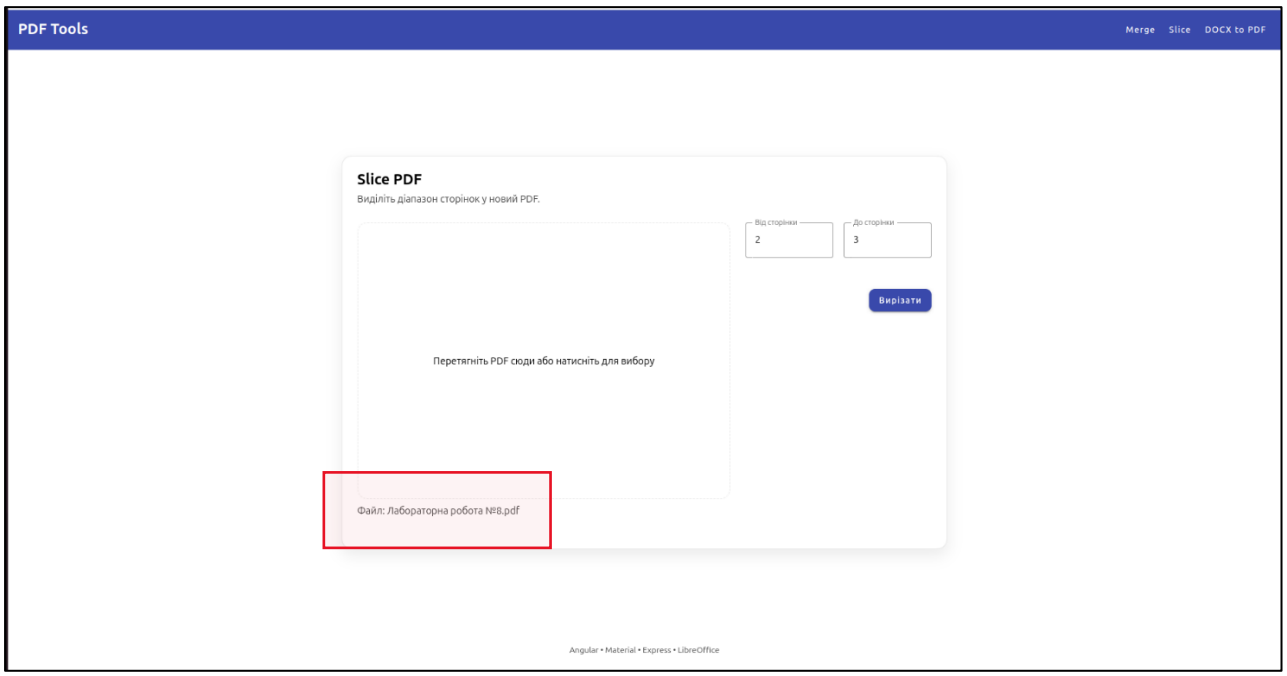



Рисунок 3.19 – Завантажений файл

Після натискання на кнопку «Вирізати» користувач отримує новий PDF лише з обраними сторінками (рис. 3.20 – 3.21).




Рисунок 3.20 – Демонстрація обробленого файлу

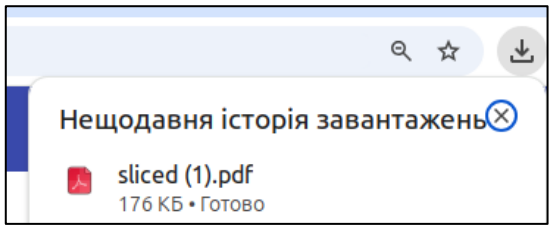



Рисунок 3.21 – Оброблений файл

### 3.6.4 Конвертація DOCX у PDF (DOCX to PDF)

Після того, як користувач переходить на сторінку DOCX to PDF, система відображає відповідний інтерфейс (рис. 3.22).

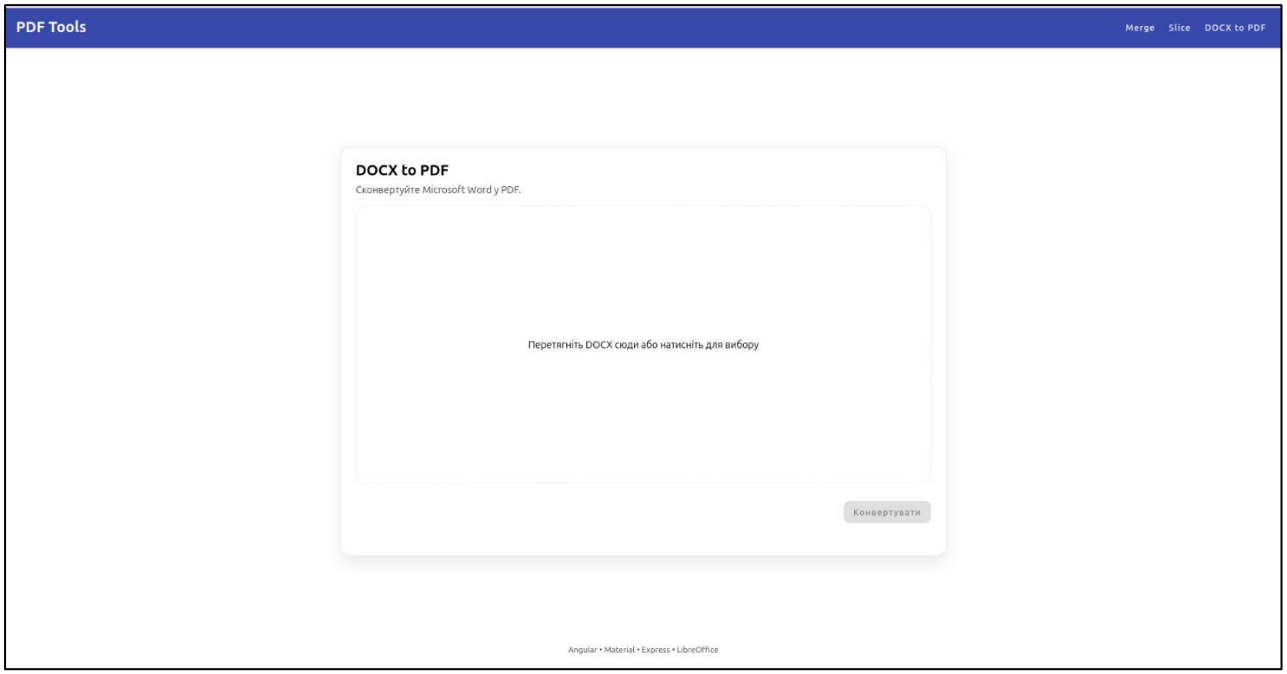



Рисунок 3.22 – Сторінка DOCX to PDF

Для активації кнопки «Конвертувати», користувачу необхідно перетягнути або обирати DOCX-файл, його назва відобразиться під зоною перетягування файлу (рис. 3.23).

Далі користувачу необхідно натиснути на кнопку «Конвертувати». Система оброблює запит та повертає отриманий PDF (рис. 3.24 – 3.25).

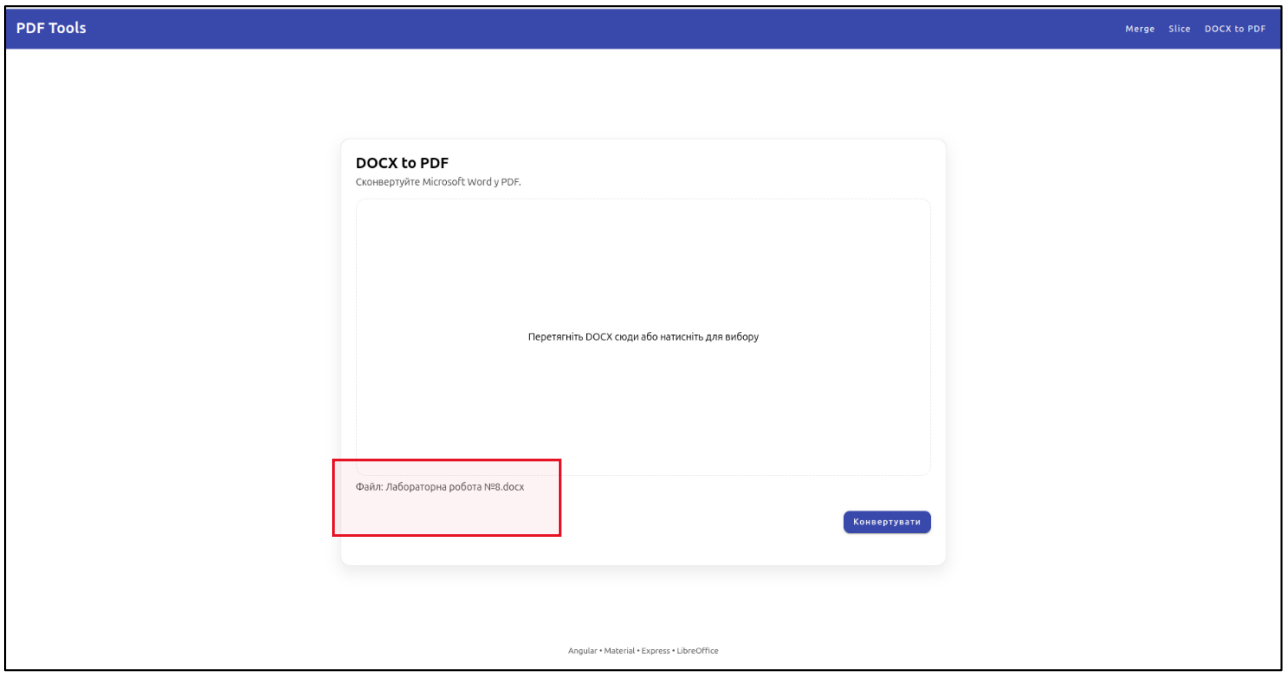



Рисунок 3.23 – Завантажений DOCX

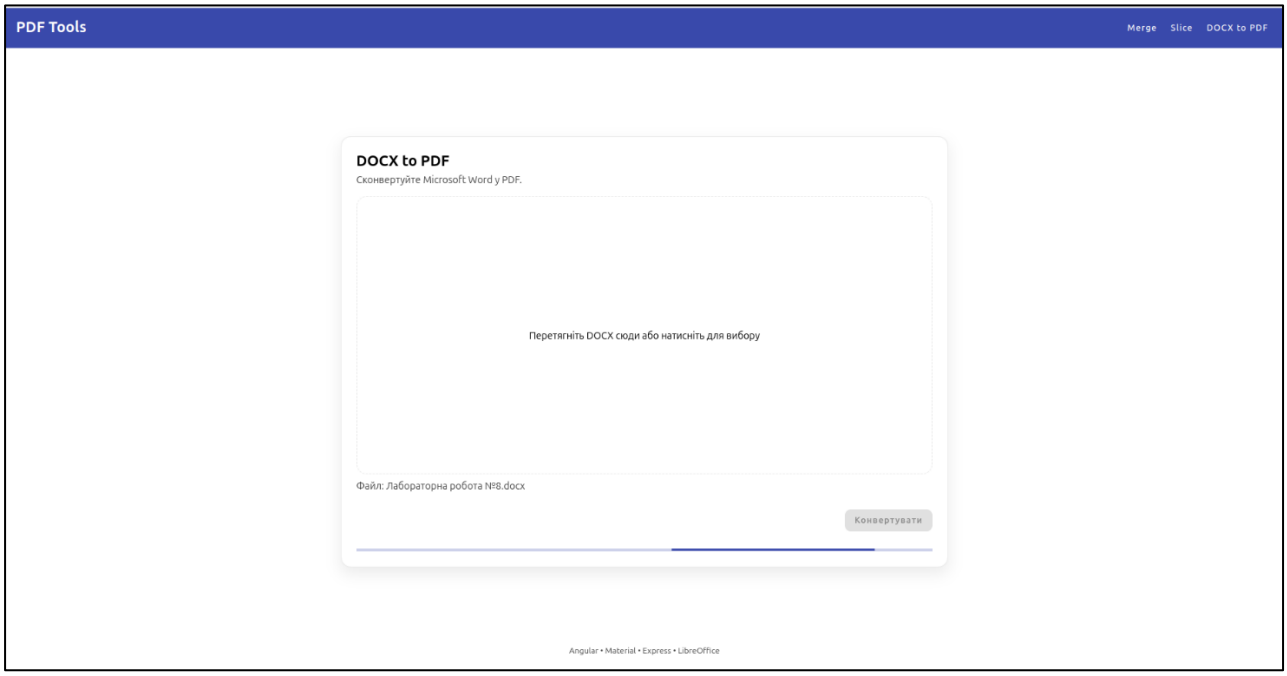



Рисунок 3.24 – Очікування закінчення процесу конвертації

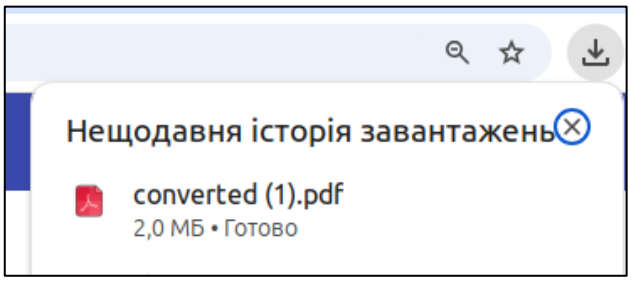



Рисунок 3.25 – Ковертований DOCX у PDF

# ВИСНОВКИ

У ході виконання роботи розроблено мікросервісний вебсервіс для базових операцій над PDF-документами. Система надає можливість швидко об’єднувати PDF, виділяти діапазони сторінок та конвертувати DOCX->PDF у браузері з мінімальною взаємодією з боку користувача. Завдяки клієнт-серверній архітектурі й модульності код легко підтримувати та масштабувати.

Відповідно до мети кваліфікаційної роботи реалізовано сервіс із використанням таких технологій:

– Angular та Material для клієнтської частини;
– Node.js та Express для серверних мікросервісів та REST API;
– Docker та Traefik для контейнеризації, маршрутизації та балансування;
– pdf-lib для операцій над PDF та LibreOffice (headless) для конвертації DOCX у PDF.

За поставленими задачами виконано такі етапи:

– сформовано вимоги до сервісу та проведено аналіз предметної області й наявних рішень;
– спроєктовано архітектуру та побудовано UML-діаграми (варіантів використання, діяльності, послідовності, класів, розгортання);
– реалізовано клієнтську частину та серверні мікросервіси з урахуванням stateless-підходу і тимчасового зберігання файлів;
– підготовлено контейнеризацію та оркестраційні налаштування (Traefik-маршрути, масштабування реплік), описано API та виконано unit-тестування ключових сценаріїв;
– надано керівництво користувача.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Adobe Acrobat Online. URL: https://www.adobe.com/ua/acrobat/online.html (дата звернення: 05.10.2025).

2. Angular Developer Documentation. URL: https://angular.io/docs/ (дата звернення: 02.10.2025).

3. Angular Material Guide. URL: https://material.angular.dev/guides (дата звернення: 02.10.2025).

4. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15. Birmingham : Packt Publishing, 2023. 446 p.

5. Chebbi L. Reactive Patterns with RxJS for Angular: A practical guide to managing your Angular application’s data reactively and efficiently using RxJS 7. Birmingham : Packt Publishing, 2022. 224 p.

6. Docker Docs. URL: https://docs.docker.com/ (дата звернення: 21.10.2025).

7. Express.js Guide. URL: https://expressjs.com/ (дата звернення: 09.10.2025).

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

9. iLovePDF. URL: https://www.ilovepdf.com/uk (дата звернення: 05.10.2025).

10. Mrzyglocki F. UML Summarized: Key Concepts and Diagrams for Software Engineers, Architects, and Designers. Phillip Mrzyglocki, 2023. 59 p.

11. Node.js Documentation. URL: https://nodejs.org/docs/latest/api/ (дата звернення: 09.10.2025).

12. npm Docs. URL: https://docs.npmjs.com/ (дата звернення: 08.10.2025).

13. npm multer. URL: https://www.npmjs.com/package/multer (дата звернення: 20.10.2025).

14. npm pdf-lib. URL: https://www.npmjs.com/package/pdf-lib (дата звернення: 21.10.2025).

15. Postman. API documentation. URL: https://www.postman.com/api-platform/api-documentation/ (дата звернення: 24.10.2025).

16. RxJS Documentation. URL: https://rxjs.dev/api (дата звернення: 24.10.2025).

17. Sejda PDF. URL: https://www.sejda.com/ (дата звернення: 05.10.2025).

18. Smallpdf. URL: https://smallpdf.com/uk (дата звернення: 05.10.2025).

19. Traefik Proxy Documentation. URL: https://doc.traefik.io/traefik/ (дата звернення: 21.10.2025).

20. Visual Paradigm. What is Unified Modeling Language (UML)?. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/ (дата звернення: 11.10.2025).

21. Visual Paradigm. Use Case Diagram Guide. URL: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення: 15.10.2025).

# ДОДАТОК А

## Angular MergeComponent

```
import { Component, HostListener } from '@angular/core';
import { ApiService } from './api.service';
import { MatButtonModule } from '@angular/material/button';
import { MatProgressBarModule } from '@angular/material/progress-bar';
import { MatSnackBar, MatSnackBarModule } from '@angular/material/snack-bar';
import { FormsModule } from '@angular/forms';
import { saveBlob } from './download.util';
import { MatCardModule } from '@angular/material/card';
import { MatIconModule } from '@angular/material/icon';
import {CommonModule} from '@angular/common';

@Component({
  standalone: true,
  selector: 'app-merge',
  imports: [MatButtonModule, MatProgressBarModule, MatSnackBarModule, FormsModule,
MatCardModule, MatIconModule, CommonModule],
  template: `
    <section class="page-full">
      <div class="panel">
        <h2>Merge PDF</h2>
        <p class="muted">Об’єднайте кілька PDF одним кліком.</p>

        <label class="dropzone" (click)="fileInput.click()" (dragover)="dragOver($event)"
(drop)="onDrop($event)">
          Перетягніть PDF сюди або натисніть для вибору
          <input #fileInput type="file" (change)="onPick($event)" accept="application/pdf" multiple hidden>
        </label>

        <div *ngIf="files.length" class="meta">Вибрано: {{files.length}} файл(и)</div>

        <div class="actions end section">
          <button mat-raised-button color="primary" (click)="submit()" [disabled]="!files.length ||
busy">Об'єднати</button>
```

```
      </div>

      <mat-progress-bar *ngIf="busy" mode="indeterminate"></mat-progress-bar>
    </div>
  </section>
 `
})
export class MergeComponent {
 files: File[] = [];
 busy = false;
 constructor(private api: ApiService, private sb: MatSnackBar) {}

 onPick(e: Event) {
  const input = e.target as HTMLInputElement;
  this.files = input.files ? Array.from(input.files) : [];
 }

 dragOver(e: DragEvent) { e.preventDefault(); (e.currentTarget as HTMLElement).classList.add('dragover');
}
 onDrop(e: DragEvent) {
  e.preventDefault(); (e.currentTarget as HTMLElement).classList.remove('dragover');
  if (e.dataTransfer?.files?.length) this.files = Array.from(e.dataTransfer.files).filter(f => f.type ===
'application/pdf');
 }
 @HostListener('dragleave', ['$event']) dragLeave(e: DragEvent) { (e.target as
HTMLElement).classList?.remove('dragover'); }

 submit() {
  this.busy = true;
  this.api.merge(this.files).subscribe({
   next: blob => saveBlob(blob, 'merged.pdf'),
   error: err => this.sb.open(err?.error?.error || 'Помилка об’єднання', 'OK', { duration: 4000 }),
   complete: () => this.busy = false
  });
 }
}
```

# ДОДАТОК Б

## Angular ApiService

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { environment } from '../environments/environment';

@Injectable({ providedIn: 'root' })
export class ApiService {
  constructor(private http: HttpClient) {}

  merge(files: File[]) {
    const fd = new FormData();
    files.forEach(f => fd.append('files', f));
    return this.http.post(`${environment.apiBase}/api/merge`, fd, { responseType: 'blob' });
  }

  slice(file: File, from: number, to: number) {
    const fd = new FormData();
    fd.append('file', file);
    fd.append('from', String(from));
    fd.append('to', String(to));
    return this.http.post(`${environment.apiBase}/api/slice`, fd, { responseType: 'blob' });
  }

  convert(file: File) {
    const fd = new FormData();
    fd.append('file', file);
    return this.http.post(`${environment.apiBase}/api/convert`, fd, { responseType: 'blob' });
  }
}
```

# ДОДАТОК В

## Express pdf.js

```
import { PDFDocument } from 'pdf-lib';

export async function mergePdf(buffers) {
  const out = await PDFDocument.create();
  for (const buf of buffers) {
    const src = await PDFDocument.load(buf);
    const pages = await out.copyPages(src, src.getPageIndices());
    pages.forEach(p => out.addPage(p));
  }
  return out.save();
}

export async function slicePdf(buffer, from, to) {
  const doc = await PDFDocument.load(buffer);
  const out = await PDFDocument.create();
  const pageCount = doc.getPageCount();
  const start = Math.max(1, Number(from));
  const end = Math.min(Number(to) || start, pageCount);
  if (!Number.isInteger(start) || !Number.isInteger(end) || start < 1 || end < start) {
    const err = new Error('Invalid page range');
    err.status = 400;
    throw err;
  }
  const pages = await out.copyPages(doc, Array.from({ length: end - start + 1 }, (_, i) => i + start - 1));
  pages.forEach(p => out.addPage(p));
  return out.save();
}
```

# ДОДАТОК Г

## Express index.js

```
import express from 'express';
import cors from 'cors';
import multer from 'multer';
import fs from 'node:fs/promises';
import os from 'node:os';
import PQueue from 'p-queue';
import { mergePdf, slicePdf } from './pdf.js';

const app = express();
app.use(cors());

const maxMb = Number(process.env.MAX_FILE_SIZE_MB || 25);
const upload = multer({ dest: os.tmpdir(), limits: { fileSize: maxMb * 1024 * 1024 } });

const mergeQ = new PQueue({ concurrency: Number(process.env.MERGE_CONCURRENCY || 2) });
const sliceQ = new PQueue({ concurrency: Number(process.env.SLICE_CONCURRENCY || 4) });

app.post('/merge', upload.array('files', 20), (req, res) => mergeQ.add(() => handleMerge(req, res)));
app.post('/slice', upload.single('file'), (req, res) => sliceQ.add(() => handleSlice(req, res)));

app.get('/health', (_req, res) => res.json({ ok: true }));

async function handleMerge(req, res) {
  try {
    if (!req.files?.length) return res.status(400).json({ error: 'Upload at least 1 PDF' });
    const buffers = await Promise.all(req.files.map(f => fs.readFile(f.path)));
    const out = await mergePdf(buffers);
    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', 'attachment; filename="merged.pdf"');
    res.send(Buffer.from(out));
  } catch (e) {
    console.error(e);
    res.status(e.status || 500).json({ error: e.message || 'Merge failed' });
```

```
  } finally {
    cleanup(req.files);
  }
}

async function handleSlice(req, res) {
  try {
    const { from, to } = req.body;
    if (!req.file) return res.status(400).json({ error: 'Upload PDF as `file`' });
    const buf = await fs.readFile(req.file.path);
    const out = await slicePdf(buf, from, to);
    res.setHeader('Content-Type', 'application/pdf');
    res.setHeader('Content-Disposition', 'attachment; filename="sliced.pdf"');
    res.send(Buffer.from(out));
  } catch (e) {
    console.error(e);
    res.status(e.status || 500).json({ error: e.message || 'Slice failed' });
  } finally {
    cleanup([req.file]);
  }
}

function cleanup(files = []) {
  files.filter(Boolean).forEach(f => fs.unlink(f.path).catch(() => {}));
}

const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`pdf-service on :${port}`));
```

# ДОДАТОК Д

## Unit-test MergeComponent

```
import { TestBed, ComponentFixture } from '@angular/core/testing';
import { By } from '@angular/platform-browser';
import { of, throwError } from 'rxjs';

import { MergeComponent } from './merge.component';
import { ApiService } from './api.service';
import { MatSnackBar, MatSnackBarModule } from '@angular/material/snack-bar';
import * as downloadUtil from './download.util';

describe('MergeComponent', () => {
  let fixture: ComponentFixture<MergeComponent>;
  let component: MergeComponent;
  let api: jasmine.SpyObj<ApiService>;
  let snack: MatSnackBar;

  beforeEach(async () => {
    api = jasmine.createSpyObj<ApiService>('ApiService', ['merge']);

    await TestBed.configureTestingModule({
      imports: [MergeComponent, MatSnackBarModule],
      providers: [{ provide: ApiService, useValue: api }],
    }).compileComponents();

    fixture = TestBed.createComponent(MergeComponent);
    component = fixture.componentInstance;
    snack = TestBed.inject(MatSnackBar);
    fixture.detectChanges();
  });

  function pdfFile(name = 'a.pdf', content = 'x'): File {
    return new File([content], name, { type: 'application/pdf' });
  }
```

```
function fileList(...files: File[]): FileList {
  // Працює у ChromeHeadless (Karma) — створюємо DataTransfer і додаємо файли
  const dt = new DataTransfer();
  files.forEach((f) => dt.items.add(f));
  return dt.files;
}

it('створюється', () => {
  expect(component).toBeTruthy();
});

it('кнопка неактивна спочатку, прогресбар відсутній', () => {
  const btn = fixture.debugElement.query(
    By.css('button[mat-raised-button]')
  ).nativeElement as HTMLButtonElement;
  expect(btn.disabled).toBeTrue();

  const bar = fixture.debugElement.query(By.css('mat-progress-bar'));
  expect(bar).toBeNull();
});

it('onPick збирає вибрані файли', () => {
  const f1 = pdfFile('a.pdf');
  const f2 = pdfFile('b.pdf');

  const input = document.createElement('input');
  Object.defineProperty(input, 'files', { value: fileList(f1, f2) });

  const ev = { target: input } as unknown as Event;
  component.onPick(ev);

  expect(component.files.length).toBe(2);
  expect(component.files[0].name).toBe('a.pdf');
  expect(component.files[1].name).toBe('b.pdf');
});

it('dragOver додає клас dragover і викликає preventDefault', () => {
  const dz = fixture.debugElement.query(By.css('.dropzone'))
```

```
    .nativeElement as HTMLElement;

  let prevented = false;
  const ev = new DragEvent('dragover', { bubbles: true });
  Object.defineProperty(ev, 'preventDefault', { value: () => (prevented = true) });
  Object.defineProperty(ev, 'currentTarget', { value: dz });

  component.dragOver(ev);
  expect(prevented).toBeTrue();
  expect(dz.classList.contains('dragover')).toBeTrue();
});

it('onDrop приймає лише PDF і очищує клас dragover', () => {
  const dz = fixture.debugElement.query(By.css('.dropzone'))
    .nativeElement as HTMLElement;

  const pdf = pdfFile('good.pdf');
  const png = new File(['x'], 'img.png', { type: 'image/png' });

  const ev = new DragEvent('drop', { bubbles: true });
  Object.defineProperty(ev, 'dataTransfer', { value: { files: fileList(pdf, png) } });
  Object.defineProperty(ev, 'preventDefault', { value: () => {} });
  Object.defineProperty(ev, 'currentTarget', { value: dz });

  dz.classList.add('dragover');
  component.onDrop(ev);

  expect(component.files.length).toBe(1);
  expect(component.files[0].name).toBe('good.pdf');
  expect(dz.classList.contains('dragover')).toBeFalse();
});




it('кнопка активується після вибору файлів', () => {
  component.files = [pdfFile()];
  fixture.detectChanges();
```

```
    const btn = fixture.debugElement.query(
      By.css('button[mat-raised-button]')
    ).nativeElement as HTMLButtonElement;

    expect(btn.disabled).toBeFalse();
  });

  it('коли busy=true, відображається прогресбар', () => {
    component.busy = true;
    fixture.detectChanges();

    const bar = fixture.debugElement.query(By.css('mat-progress-bar'));
    expect(bar).toBeTruthy();
  });
});
```