

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет

Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

«До захисту допущено»

В.о. завідувача кафедри

Оксана ШОВКОПЛЯС

(підпис)

грудня 2024 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 «Комп'ютерні науки»

освітньо-професійної програми «Інформатика»

на тему: Інтелектуальна технологія оцінки доступності офісних документів

здобувача групи ІН.м - 32 Зулюкова Івана Юрійовича

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

Іван ЗУЛЮКОВ

(підпис)

Керівник,
доцент кафедри КН,
к.т.н., доцент

Борис КУЗІКОВ

(підпис)

Суми – 2024

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»

В.о. завідувача кафедри

Оксана ШОВКОПЛЯС

(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.М-32 Зулюкова Івана Юрійовича

1. Тема роботи: «Інтелектуальна технологія оцінки доступності офісних документів»
затверджую наказом по СумДУ від «03» грудня 2024 року №1257-VI
2. Термін здачі здобувачем кваліфікаційної роботи до 06 грудня 2024 року
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для оцінки доступності документів
3) Розробка інтелектуальної технології оцінки доступності офісних документів
4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
6. Консультанти до проекту (роботи), із зазначенням розділів проекту, що стосується їх _____

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання « ____ » _____ 20 ____ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	<i>Аналіз проблеми предметної області, постановка й формування завдань дослідження</i>		
2	<i>Огляд технологій, що використовуються для оцінки доступності документів</i>		
3	<i>Розробка інтелектуальної технології оцінки доступності офісних документів</i>		
4	<i>Аналіз отриманих результатів</i>		
5	<i>Оформлення пояснювальної записки до кваліфікаційної роботи</i>		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 71 стр., 24 рис., 1 табл., 1 додаток, 35 використаних джерел.

Обґрунтування актуальності теми роботи – Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої практичної задачі оцінки доступності офісних документів шляхом розробки відповідної інтелектуальної технології.

Об’єкт дослідження — процес забезпечення доступності електронних документів.

Мета роботи — розробка інтелектуальної технології оцінки доступності документів у форматі PDF.

Методи дослідження — моделі штучного інтелекту та інструменти перевірки відповідності стандартам доступності

Результати — розроблено інтелектуальну технологію для аналізу доступності документів, що базується на застосуванні ряду аналізаторів, специфічних до формату документу, та забезпечує агрегацію та гармонізацію отриманих звітів із використанням засобів штучного інтелекту. Технологія реалізована у вигляді веб сервісу, створеного за допомогою фреймворку React з використанням бібліотеки Material UI, який дозволяє проаналізувати PDF документи із допомогою VeraPDF REST API та VeraPDF Web Server API та обробити звіт за допомогою великих мовних моделей Qwen, Mixtral, Llama та Gemma сервісу Hugging Face із використанням Serverless Inference API. Проведено тестування розробки на файлах що відповідають стандартам доступності та файлі що не відповідає стандартам доступності.

PDF, JAVASCRIPT, REACT, MUI, VERAPDF, LLM, HUGGING FACE, API.

ЗМІСТ

ВСТУП.....	5
1 АНАЛІТИЧНИЙ ОГЛЯД	6
1.1 Сучасний стан доступності офісних документів	6
1.2 Огляд наявних рішень оцінки та покращення доступності документів у форматі PDF	17
1.3 Постановка задачі	31
2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ.....	32
2.1 Вибір мови програмування.....	32
2.2 Вибір API.....	33
2.3 Вибір бібліотек	35
2.4 Додаткові інструменти.....	36
2.5 Проєктування інтелектуальної технології	38
3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ	44
3.1 Формування вхідних даних	44
3.2 Опис програмної реалізації	44
3.3 Аналіз результатів	48
ВИСНОВКИ.....	53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	54
ДОДАТОК А ЛІСТИНГ ПРОГРАМИ.....	58

ВСТУП

Актуальність. У сучасному світі оцінка доступності офісних документів є важливою складовою рівного доступу до інформації. Інтелектуальні технології здатні зменшити кількість помилок і спростити перевірку відповідності документів стандартам доступності, що безпосередньо впливає на продуктивність і комфорт користувачів. Запровадження таких рішень відкриває нові можливості для автоматизації роботи, підвищення якості обслуговування та дотримання нормативних вимог. Розробка інтелектуальної технології оцінки доступності офісних документів сприятиме покращенню доступу до інформації та матиме попит серед бізнесу, державних установ і освітніх організацій.

Об'єкт дослідження. Процес забезпечення доступності електронних документів.

Предмет дослідження. Технології та методи оцінки та підвищення доступності офісних документів.

Новизна. На відміну від існуючих інструментів аналізу доступності офісних документів, описане у даній роботі програмне рішення дозволить перевірити відповідність документу обширному списку стандартів доступності та представити результат у зрозумілому вигляді завдяки вичерпним поясненням важливості кожної знайденої проблеми та шляхами їх вирішення.

Структура. Робота складається зі вступу, аналітичного огляду, постановки задачі, вибору методу розв'язання поставленої задачі, опису програмного забезпечення інтелектуальної технології, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан доступності офісних документів

Забезпечення доступності офісних документів є критично важливим завданням, особливо в умовах, коли інтернет значно розширив межі комунікації, навчання та робочих процесів. Доступні документи забезпечують рівний доступ до інформації для всіх користувачів, зокрема для осіб з інвалідністю, що є важливою складовою інклюзивного суспільства. У сучасних організаціях та бізнес-середовищі електронні документи використовуються як основний засіб обміну інформацією, управління знаннями та прийняття рішень. Забезпечення доступності цих документів сприяє продуктивності, покращує ефективність роботи та допомагає дотримуватися нормативних вимог і стандартів.

Особливо важливим питання доступності документів є в державному секторі, освітніх установах та великих організаціях, де цифрова доступність регламентована законами і стандартами. Порушення доступності може спричинити обмеження у правах на інформацію для деяких категорій населення, зокрема для осіб з порушенням зору чи слуху, а також для людей, які використовують спеціальні програми для читання текстів.

Різні формати офісних документів, як-от текстові файли (DOC, DOCX), таблиці (XLS, XLSX) чи презентації (PPT, PPTX), мають свої особливості забезпечення доступності. Водночас формат PDF (Portable Document Format) є одним із найбільш популярних для зберігання, архівування та обміну документацією через низку суттєвих переваг. PDF/A дозволяє зберегти цілісність та стабільність оформлення документа незалежно від пристрою або програмного забезпечення, що його відкриває. Це особливо важливо в ситуаціях, де важлива точна передача візуального вигляду або коли документи розповсюджуються для публічного доступу. Крім того, PDF-документи здатні інтегрувати текст, зображення, таблиці та інші елементи, що робить цей формат універсальним для різних сфер застосування — від бізнесу до освітніх і державних потреб.

Проте, хоча PDF і є зручним для збереження та розповсюдження інформації, цей формат має статичну природу. Це створює певні труднощі при адаптації документа для людей з особливими потребами, оскільки вміст у PDF може бути представленим неструктурованим, що ускладнює його читання та взаємодію зі зчитувачами з екрану. Враховуючи зростаючу кількість PDF-документів у відкритому доступі, питання їхньої доступності стає надзвичайно актуальним.

Для забезпечення доступності PDF-документів розроблено низку стандартів, які допомагають зробити такі документи придатними для використання різними категоріями користувачів.

WCAG

В січні 1995 року Грег Вандерхайден опублікував першу настанову щодо веб доступності. Більше 38-и інших настанов було опубліковано різними авторами в наступні роки [1] та згодом об'єднано в уніфіковані настанови доступності веб-сайтів(UWSAG). 20 січня 1998 року було опубліковано восьму версію UWSAG [2] що стало відправною точкою для створення WCAG 1.0.

5 травня 1999 року було опубліковано та рекомендовано першу версію настанов щодо доступності веб-вмісту(WCAG 1.0) яка містила 14 настанов.

Кожна настанова охоплює основну тему веб-доступності та пов'язана з одним або кількома критеріями успішності, які описують, як застосувати цю настанову до певних функцій веб-сторінки. Кожен із загалом 65 критеріїв успішності WCAG 1.0 має призначений рівень відповідності на основі впливу критерію успішності на доступність:

- Рівень відповідності А: Веб-розробники повинні задовольняти ці вимоги, інакше одна чи декілька груп не зможуть отримати доступ до веб-вмісту.
- Рівень відповідності АА: Веб-розробникам слід задовольняти ці вимоги, інакше деяким групам буде важко отримати доступ до веб-вмісту.

- Рівень відповідності AAA: Веб-розробники можуть задовольнити ці вимоги, щоб полегшити деяким групам доступ до веб-вмісту.

25 січня 2001 року було опубліковано першу пропозицію WCAG 2.0 але через велику кількість поправок рекомендовано цю версію було тільки 11 грудня 2008 року [3]. 12 жовтня 2012 року WCAG 2.0 став міжнародним стандартом ISO/IEC 40500 [4]. Версія 2.0 має лише 12 настанов на відміну від чотирнадцяти версії 1.0. Кількість критеріїв успішності також зменшилась з 65 до 61. Більш того настанови було організовано за чотирма принципами: веб-сайти повинні бути придатні до сприйняття, керовані, зрозумілі та надійні. Рівні відповідності версія 2.0 використовує ті ж самі що й версія 1.0 але їх визначення було змінено.

21 вересня 2023 року було опубліковано та рекомендовано WCAG 2.1 [5]. В новій версії було додано 17 критеріїв успішності піднявши загальну кількість до 78.

5 жовтня 2023 року було опубліковано та рекомендовано WCAG 2.2 [6]. В цій версії було додано ще 9 критеріїв успішності але один з попередніх 78 критеріїв успішності було видалено що змінило загальну кількість критеріїв до 86.

PDF/A

28 вересня 2005 року було опубліковано PDF/A-1 (ISO 19005-1) [7] створений на основі PDF 1.4. Головною метою була стандартизація архівації електронних документів. Стандарт поділяється на два рівні відповідності: Базовий(B) та Доступний(A).

Доступний рівень включає в себе всі правила Базового та включає додаткові правила для забезпечення доступності, зокрема:

- Специфікація мови
- Ієрархічна структура документа
- Позначений тегами текст охоплює та описовий текст для зображень і символів
- Відображення символів у Юнікод

20 червня 2011 року було опубліковано PDF/A-2 (ISO 19005-2) [8] створений на основі PDF 1.7 (ISO 32000-1:2008). З переходом на більш сучасну PDF специфікацію стандарт отримав ряд нових можливостей. Окрім рівнів відповідних В та А попередньої частини новий стандарт також отримав рівень Юнікод (U) що відповідає Базовому але подібно Доступному потребує відображення всього тексту у символах юнікоду.

15 жовтня 2012 року було опубліковано PDF/A-3 (ISO 19005-3) [9] створений на основі PDF 1.7 (ISO 32000-1:2008). Єдиною відмінністю від попередньої частини стала можливість вбудування файлів в PDF документ.

В листопаді 2020 року було опубліковано PDF/A-4 (ISO 19005-4) [10] створений на основі PDF 2.0 (ISO 32000-2:2020). Даний стандарт було створено на заміну PDF/E для архівації інженерних електронних документів. Новий Базовий рівень замінив рівні В та U. Також було додано Файловий (F) рівень що дозволяє вбудову сторонніх файлів в документ та Інженерний (E) рівень що розширює Файловий рівень підтримкою 3D, RichMedia та JavaScript. Рівень А не було реалізовано так як для цього був створений стандарт PDF/UA.

PDF/UA

15 липня 2012 було опубліковано PDF/UA-1 (ISO 14289-1:2012) [11] створений на основі PDF 1.7 (ISO 32000-1:2008). Метою даного стандарту було виправлення недоліків Доступного рівня PDF/A-2.

18 грудня 2014 було опубліковано оновлену версію PDF/UA-1 (ISO 14289-1:2014) [12]. Оскільки стандарти Міжнародної Організації зі Стандартизації(ISO) поширюються у форматі PDF ця версія стала першим повноцінно доступним ISO стандартом так як сам документ відповідає стандарту PDF/UA.

Головною ціллю даного стандарту було забезпечення доступності PDF документів шляхом регуляції всього життєвого шляху документу. Пункт 7 регулює створення документу за допомогою наступних вимог:

- Повне тегування «справжнього вмісту» в логічному порядку читання

- Теги повинні правильно відображати семантичні структури документа (заголовки, списки, таблиці тощо)
- Проблемний вміст заборонено, зокрема нелогічні заголовки, використання кольору/контрасту для передачі інформації, недоступний JavaScript тощо
- Змістовна графіка повинна містити альтернативні текстові описи
- Параметри безпеки мають дозволяти доступ допоміжних технологій до вмісту
- Шрифти мають бути вбудовані, а текст зіставлений із Unicode

Пункт 8 регулює зчитування та взаємодію з документом через процесор за допомогою наступних вимог:

- Можливість повної обробки тегів і артефактів у файлах PDF
- Можливість повідомляти користувачам про відповідні дії
- Можливість обробки та представлення цифрових підписів, анотацій і додаткового вмісту
- Можливість навігації в документі різними засобами

Пункт 9 регулює функціонал допоміжних технологій:

- Здатність представляти користувачеві АТ інформацію, надану PDF/UA-зчитувачем
- Можливість навігації в документі різними засобами

Хоча специфікація PDF/UA написана для розробників програмного забезпечення, підтримка PDF/UA цікавить людей з обмеженими можливостями, які потребують або отримують переваги від допоміжних технологій під час читання електронного вмісту. Завдяки файлам, що відповідають формату PDF/UA, програмам для читання та допоміжним технологіям, користувачам гарантується рівний доступ до інформації, наскільки це може забезпечити сам формат PDF.

Переваги PDF/UA виходять не тільки для людей з обмеженими можливостями. Завдяки підтримці PDF/UA програмне забезпечення для читання

зможе надійно перекомпоновувати текст на маленьких екранах, надавати потужні можливості навігації, змінювати вигляд тексту, покращувати функціональність пошукової системи, допомагати у виборі та копіюванні тексту тощо.

В 2017 році було опубліковано стандарт PDF 2.0 (ISO 32000-2). В 2020 році цей стандарт було оновлено та значно покращено специфікацію Tagged PDF що є головним механізмом забезпечення доступності PDF файлів. В лютому 2024 PDF асоціацією було опубліковано стандарт Well Tagged PDF(WTPDF) в якому були визначені вимоги щодо доступності файлів PDF 2.0.

15 березня 2024 було опубліковано PDF/UA-2 (ISO 14289-2:2024) [13] створений на основі WTPDF в свою чергу створений на основі PDF 2.0 (ISO 32000-2:2020).

Окрім надання правил доступності, специфічних для файлів PDF 2.0, у порівнянні з ISO 14289-1 новий стандарт ISO містить низку вдосконалень основних вимог до Tagged PDF, зокрема:

- комплексні вимоги до атрибутів елементів структури;
- комплексні вимоги до використання анотацій;
- приклади використання семантично значущих атрибутів.

WTPDF і PDF/UA-2 вводять інші вдосконалення вимог до доступності, використовуючи вдосконалення основного формату PDF 2.0, зокрема:

- комплексні вимоги до нових типів структурних елементів, представлених у PDF 2.0;
- правила, що регулюють включення структурних елементів, визначених у PDF 1.7, а також у PDF 2.0;
- детальні вимоги до математики, включаючи підтримку MathML;
- вимоги до внутрішньодокументних посилань із використанням функції призначення структури;
- використання функції пов'язаних файлів для полегшення інтеграції вмісту, відмінного від PDF.

Порівняння PDF/UA з WCAG

Головним недоліком PDF/UA є його ціна. ISO 14289-1:2014 коштує 96 швейцарських франків [14] тоді як ISO 14289-2:2024 коштує аж 173 швейцарських франки [15]. В серпні 2013 року для полегшення доступу до PDF/UA-1 зусиллями PDF асоціації було розроблено Matterhorn Protocol – збірник з 31 критерію успішності та 136 шляхів їх провалити. PDF асоціація також завіряє що документ який відповідає стандарту WTPDF також відповідає стандарту PDF/UA-2. Більш того 12 серпня 2024 року завдяки допомозі чотирьох компаній спонсорів PDF асоціація дозволяє безкоштовно придбати обидві версії PDF/UA. Тим не менш не відомо чи є ця пропозиція безстроковою.

З іншого боку головним недоліком WCAG є його абстрактність адже він є доволі застарілим та був розроблений для забезпечення доступності веб-контенту а не тільки PDF файлів.

Хоча прийнято вважати що відповідність стандарту PDF/UA означає відповідність стандарту WCAG технічно це не зовсім вірно. На наступній діаграмі(див. Рисунок 1.1) можна побачити порівняння PDF/UA-1 з WCAG 2.0:

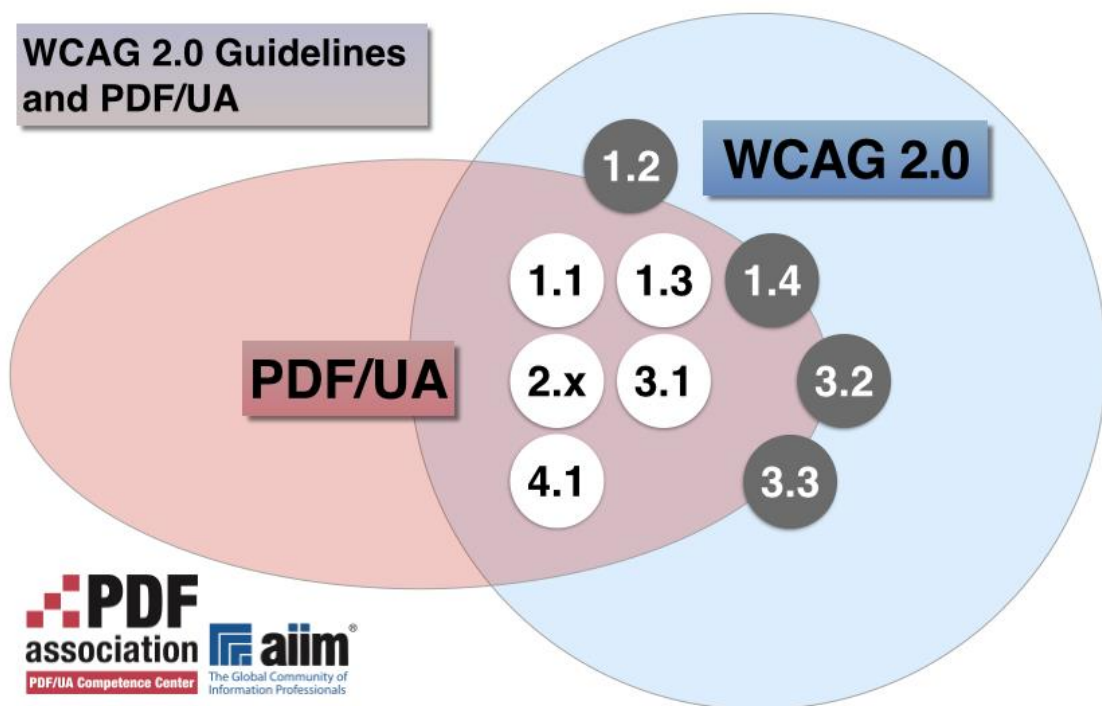


Рисунок 1.1 - Діаграма відношення PDF/UA-1 до WCAG 2.0 [16]

На діаграмі показано, що хоча WCAG 2.0 і PDF/UA охоплюють суттєво різні сфери, між ними існує значне перехрестя за настановами WCAG 2.0. WCAG 2.0 є загальним і високорівневим стандартом, особливо в частинах «Принципи» і (меншою мірою) «Настанови», тоді як найбільше конкретики зосереджено у критеріях успішності.

PDF/UA охоплює вісім настанов WCAG 2.0, а чотири з них враховані частково. Частково охопленими є такі настанови:

- 1.2 Мультимедійний матеріал, обмежений за часом,
- 1.4 Розрізнення,
- 3.2 Передбачуваність,
- 3.3 Допомога під час введення.

Більшість відмінностей між стандартами стосується лише специфічних типів контенту.

PDF/UA фокусується на форматі PDF-документів і лише побічно охоплює сторонній контент, який може бути включений у PDF, але зазвичай не використовується.

Відповідно, PDF/UA включає мінімальні нормативні вимоги для мультимедійного матеріалу, обмеженого за часом, обмежуючись текстовим описом для мультимедійних файлів.

Для програмного забезпечення PDF/UA вимагає наявність доступних елементів керування (за їхньої наявності) і забороняє автоматичне відтворення.

Наступна діаграма (див. Рисунок 1.2) показує, що в PDF/UA частково враховано лише критерій успішності 1.2.3.

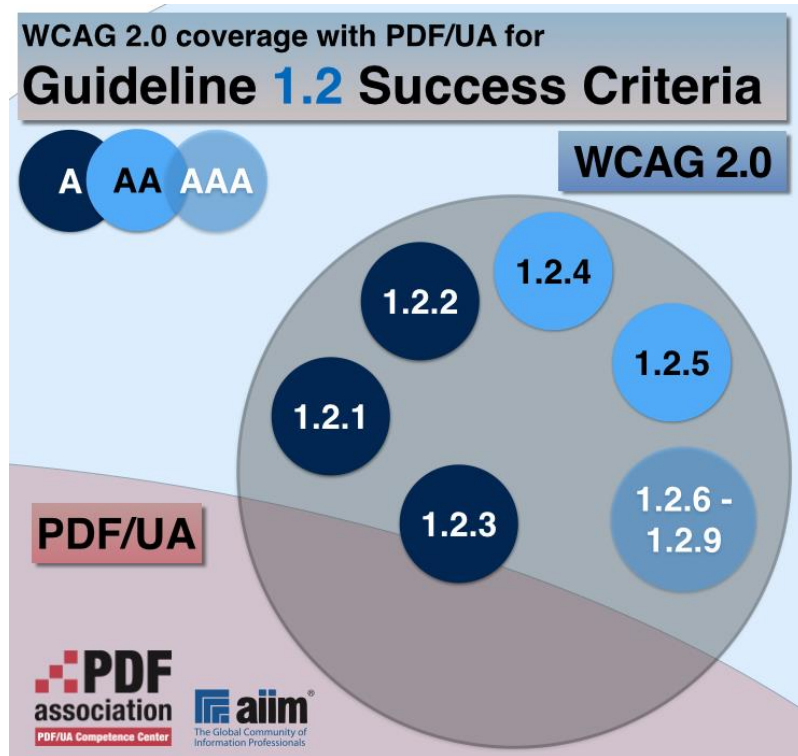


Рисунок 1.2 - Діаграма відповідності PDF/UA з критеріями успішності для настанови 1.2 WCAG 2.0 [16]

WCAG 2.0 встановлює критерії успішності для оцінки доступності вебконтенту, тоді як PDF/UA визначає вимоги до оцінки доступності у використанні формату PDF.

Ця відмінність є ключовою для розуміння різних підходів стандартів до питання розрізнення контенту. PDF/UA фокусується на забезпеченні доступності інформації через функцію тегування PDF, тоді як WCAG 2.0 припускає більший контроль над виглядом кожного елемента контенту.

Наступна діаграма (див. Рисунок 1.3) показує, що критерії успішності 1.4.1, 1.4.2 і 1.4.4 повністю враховані в PDF/UA. Критерії успішності 1.4.3 і 1.4.5–1.4.9 не враховані в PDF/UA.

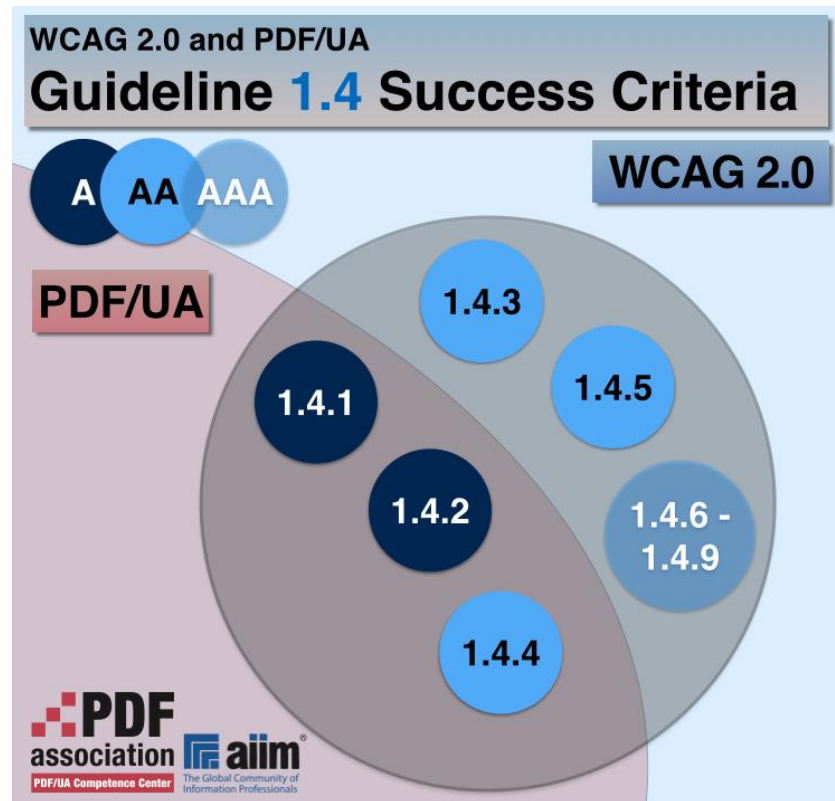


Рисунок 1.3 - Діаграма відповідності PDF/UA з критеріями успішності для настанови 1.4 WCAG 2.0 [16]

Більшість положень у настановах 3.2 і 3.3 стосуються роботи зі скриптами. JavaScript часто використовується в PDF-документах, особливо в формах, для забезпечення різних функцій навігації, логіки та підвищення доступності.

Однак PDF/UA залишається специфічним для PDF-формату, а вимоги до JavaScript у PDF/UA є вузько визначеними, особливо порівняно з настановами 3.2 і 3.3 у WCAG 2.0.

Наступна діаграма (див. Рисунок 1.4) показує, що критерії успішності 3.2.1 і 3.2.2 частково враховані у PDF/UA, тоді як критерії 3.2.3–3.2.5 не враховані.

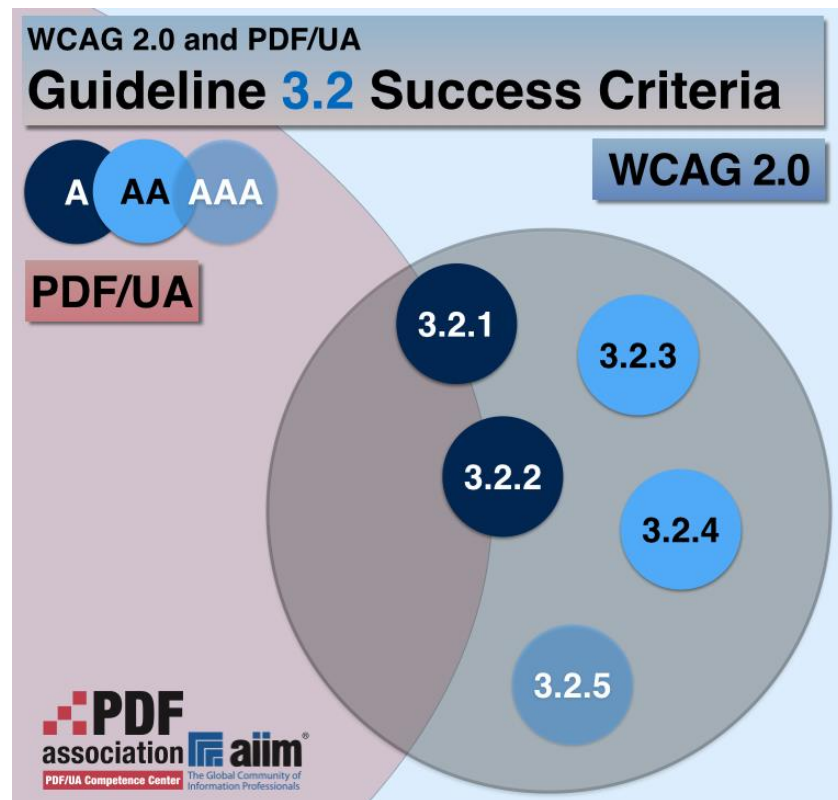


Рисунок 1.4 - Діаграма відповідності PDF/UA з критеріями успішності для настанови 3.2 WCAG 2.0 [16]

Для настанови 3.3 більшість вимог з точки зору PDF залишається на розсуд автора під час створення документа, тому вони не є специфічними для PDF.

Загалом PDF/UA не охоплює візуальний вигляд, контент або вибір дизайну і покладається на семантичні еквіваленти. У таких випадках PDF/UA зазвичай посилається на WCAG 2.0 як нормативно, так і інформативно.

Наступна діаграма(див. Рисунок 1.5) показує, що критерій успішності 3.3.2 повністю врахований у PDF/UA, але критерії успішності 3.3.1 і 3.3.3–3.3.6 не враховані.

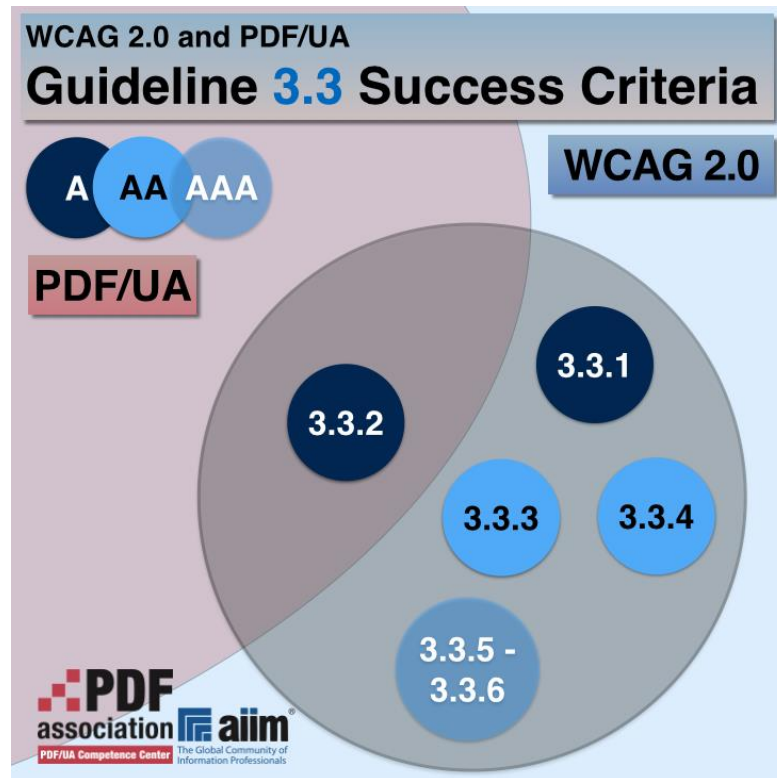


Рисунок 1.5 - Діаграма відповідності PDF/UA з критеріями успішності для настанови 3.2 WCAG 2.0 [16]

1.2 Огляд наявних рішень оцінки та покращення доступності документів у форматі PDF

Adobe Accessibility Checker

Adobe Acrobat є одним із найпопулярніших і найпоширеніших програмних продуктів для роботи з PDF-документами. Програма надає користувачам широкий спектр інструментів для створення, редагування, перегляду та захисту PDF-файлів, а також інтеграції з іншими додатками та платформами. Завдяки своїй функціональності Adobe Acrobat став стандартним інструментом для організацій, які працюють із цифровими документами, включаючи компанії, освітні установи та державні органи. Значна популярність цього інструменту спонукає користувачів звертатися до вбудованих можливостей перевірки доступності.

Переваги:

- Можливість виправити недолік доступності в редакторі

- Можливість знайти місцезнаходження недоліку доступності в документі

Недоліки:

- Інструмент доступний лише в Acrobat Pro що розповсюджується на комерційній основі у вигляді підписки що коштує 30\$ в місяць [17]
- Неповна підтримка PDF/UA

Adobe також тестує REST API для цього інструменту але доступ до тестування поки що закритий [18]. З іншого боку використання цього інструменту за допомогою API унеможлиблює відображення панелі перевірки доступності що позбуває цей інструмент його головних переваг.

allyant CommonLook

Якщо користувачу не до вподоби вбудований в Adobe Acrobat Pro інструмент компанія allyant пропонує пакет інструментів CommonLook що включає [19]:

CommonLook PDF Validator. Цей безкоштовний плагін для Adobe Acrobat перевіряє PDF-файли на відповідність численним стандартам доступності, зокрема Section 508, WCAG AA, PDF/UA та настановам HHS. Він точно визначає проблеми доступності, надаючи безцінне розуміння проблем відповідності.

CommonLook PDF. Розширений плагін Adobe Acrobat із оптимізованими, економічно ефективними та надійними методами для досягнення відповідності стандартам доступності WCAG, PDF/UA та Section 508. Цей надійний інструмент спрощує шлях до відповідності доступності, яка визнана в техніках PDF W3C для WCAG.

CommonLook Office. Цей інструмент створює доступні PDF-файли з документів Microsoft Word і PowerPoint. Він надає зручний посібник для створення PDF-файлів, які відповідають різним стандартам доступності, що полегшує відповідність вимогам.

CommonLook Online. Це програмне забезпечення для виправлення документів, розроблене для професіоналів у сфері освіти, забезпечує швидке,

ефективне та рентабельне транскрибування доступних документів. Воно гармонізує вдосконалену автоматизацію зі зручними інструкціями для підтримки якості та можливостей переформатування, перетворення документів у різні доступні формати.

CommonLook Clarity. Сканує, відстежує та забезпечує доступність PDF-файлів на цифрових платформах, допомагаючи підтримувати відповідність стандартам доступності.

Accessible PDF Automation Solutions. Ефективно створює доступні PDF-файли у великому обсязі для шаблонних документів, забезпечуючи узгодженість і відповідність усіх стандартизованих ресурсів, форм і звітів.

На жаль доступ до цих інструментів закритий та потребує зв'язку з представниками компанії для придбання.

Переваги:

- Підтримка великої кількості стандартів
- Можливість інтеграції в Adobe Acrobat та Microsoft Office
- Можливість автоматизації

Недоліки:

- Неінтуїтивний шлях завантаження безкоштовних продуктів
- Невизначена ціна комерційних продуктів

Adobe PDF Accessibility Auto-Tag

Ще один інструмент доступний в Acrobat Pro дозволяє автоматично тегувати PDF документ. Тегування не вносить візуальних змін в документ але додає додаткову інформацію про зміст документу що використовується допоміжними технологіями для озвучування вмісту документу.

Переваги:

- Можливість автоматичного тегування документу що є основою його доступності
- Можливість виправлення неправильно тегованих елементів вручну

Недоліки:

- Низька якість автоматичного тегування
- Відсутність відповідності жодному зі стандартів
- Інструмент доступний лише в Acrobat Pro що розповсюджується на комерційній основі у вигляді підписки що коштує 30\$ в місяць

Adobe також надає можливість використання даного інструменту через REST API та на відміну від Accessibility Checker API знаходиться у відкритому доступі. На відміну від використання вбудованого в Acrobat Pro інструменту використання API позбавляє користувача можливості одразу перевірити та виправити неправильно згенеровані теги. На жаль використання обмежене 500 запитами на місяць [20].

PAVE

Даний інструмент було розроблено лабораторією ICT-Accessibility Lab Цюрихського університету прикладних наук (ZHAW) у співпраці зі Швейцарською федерацією сліпих та людей із вадами зору [21].

Ціллю даного інструмента є вирішення тієї самої проблеми що вирішує Adobe Accessibility Auto-Tag але на відміну від нього цей інструмент було розроблено на 10 років раніше та його використання є необмеженим. Більш того інструмент надає перелік проблем що потрібно виправити вручну(див. Рисунок 1.6) та надає можливість зробити це через вікно результатів(див. Рисунок 1.7).

Переваги:

- Необмежений безкоштовний доступ
- Можливість редагування тегів вручну

Недоліки:

- Низька якість автоматичного тегування
- Відсутність відповідності жодному зі стандартів

TASKS **PROPERTIES** **ISSUE DETAILS** **READING ORDER**

Automatic Correction

PAVE was already able to automatically correct 82 issues.

Title missing

The document's properties contain information such as the document's title and language. Some of these are mandatory. To fix these issues, correct the fields in the document properties.

[EDIT PROPERTIES](#)

Some tags are incorrectly nested

To allow screenreaders to correctly read the document, not only do all elements (texts, images, etc.) have to be tagged correctly. They also have to be ordered in the correct reading order. Individual issues for each page are listed in the tab **ISSUE DETAILS**. To add and fix tags, select elements in the document that should be tagged. You can find an overview of the different tags in the [FAQ](#).

[SHOW ISSUE DETAILS](#)

Рисунок 1.6 - Перелік проблем що потрібно виправити вручну

Tagging CANCEL

- 1 Select element type**
Element type
Other
- 2 Select tag**
This is for advanced users. Select the specific tag to apply.
Tag
Link
- 3 Select content language (optional)**
If the selected content has the same language as the document, leave this empty. Otherwise, choose the language used in the selected content.
Language
- 4 Enter alternative text**
Enter a text that describes the content. The text should provide the reader with the same information as viewing the content would. Note that you can use multiple lines to break up the text.

Рисунок 1.7 - Вікно додання тегу вручну

callas software pdfaPilot

Хоча Accessibility Checker і вбудований в Adobe Acrobat Pro його розробкою займалася не Adobe а компанія callas software. Зважаючи на те що тестування REST API для Accessibility Checker почалося лиш в 2024 році та ще досі є закритим доволі довгий час цей інструмент не мав ніякого програмного інтерфейсу. Для заповнення цієї ніші callas software створили pdfaPilot.

pdfaPilot — це рішення для перетворення документів PDF у надійні файли PDF/A з можливістю пошуку для тривалого архівування. Він використовує ту саму технологію PDF/A, що й у Adobe Acrobat, і доступний для використання на робочому столі та готовий для інтеграції в існуючі системи за допомогою інтерфейсу командного рядка чи набору програмного забезпечення [22].

pdfaPilot перевіряє більшість вимог PDF/UA які можна перевірити програмним забезпеченням. Для перевірки людиною інтегровані функції для перегляду документа в структурованому та виділеному веб-перегляді. Після перевірки можна створити повний звіт про відповідність.

Переваги:

- Можливість виправлення файлу вручну в Desktop версії
- Можливість автоматизації за допомогою Server, CLI та SDK версій

Недоліки:

- Desktop версія коштує 625 євро [23]
- Набір Server + CLI коштує 6000 євро [24]

ЕІІ Tingtun Checker

В 2017 році Європейська Ініціатива Інклюзивності Інтернету(ЕІІ) [25] розробила CLI додаток PDFWAM що перевіряє доступність PDF файлів за допомогою 14 технік WCAG 2.0 [26].

Варто зазначити що покриття WCAG 2.0 неповне адже інструмент не застосовує 9 з 23 наявних на той час технік [27].

Техніки W3C для WCAG 2.0 — це список технік, які допомагають авторам відповідати вказівкам і критеріям успіху. Методики періодично оновлюються, тоді як принципи, рекомендації та критерії успіху є стабільними та не змінюються. Результатом виконання(див. Рисунок 1.8) є невеличкий звіт з таблицею виконаних тестів, результатом їх виконання та коротким описом кожного з тестів.

```

***Test Report***
-----
TEST | STATUS | DESCRIPTION
-----
wcag.pdf.03 | Pass | tab and reading order
egovmon.pdf.03 | Pass | structure tree
wcag.pdf.14 | Fail | running headers/footers
wcag.pdf.16 | Pass | natural language
egovmon.pdf.05 | Pass | permissions
wcag.pdf.02 | Pass | bookmarks
egovmon.pdf.08 | Pass | scanned
wcag.pdf.18 | Fail | title
-----
Test summary: 8 total tests, 2 fail, 6 pass

```

Рисунок 1.8 - Результат виконання PDFWAM

Для полегшення використання інструменту було створено сайт під назвою Tingtun Checker що дозволяє завантажити або надати посилання на PDF файл для перевірки [28]. На жаль в 2023 році сайт почав видавати помилку Server Error 500 при спробі перевірити файл що унеможливило його використання. Вирішенням цієї проблеми зайнялась компанія Cantan Group. Використовуючи PDFWAM функціонал сайту було відновлено та вбудовано в пакет інструментів ACSHECKS [29]. Використавши відновлений функціонал можна побачити що сайт дозволяє не лише полегшити доступ до використання PDFWAM але й надає набагато зрозуміліший звіт. Як можна побачити на нижче наведеному знімку екрану(див. Рисунок 1.9) кожен тест відображений у вигляді картки з назвою тесту, коротким описом та посиланням на техніку WCAG 2.0 на основі котрої було створено тест.

Переваги:

- Зрозумілий опис кожної з проблем
- Наданий шлях рішення для кожної проблеми

Недоліки:

- Використання застарілого стандарту WCAG 2.0 та технік 2012 року
- Неповне покриття WCAG 2.0

Structure Elements (tags) Document should have structure elements (tags) EXPLANATION Pass	Document Permissions Document permissions should allow assistive technology to access the PDF content. EXPLANATION Pass	Scanned Document Document should have textual content. EXPLANATION Pass	Alternative Text for Images Document should have alternative text for images. EXPLANATION N/A
Bookmarks Document should have bookmarks. EXPLANATION Pass	Correct Tab and Reading Order Document should specify reading order that is consistent with the logical structure. EXPLANATION Pass	Decorative Images Decorative images should be marked in such a way that they can be ignored by assistive technology. EXPLANATION N/A	Table Elements Table elements should use the correct structure to help determine the context of a table entry such as the row and column titles. EXPLANATION N/A
Heading Levels Document should have consistent heading levels. EXPLANATION N/A	Forms Fields (name/role/value) Document should have correctly marked up form fields. EXPLANATION N/A	Running Headers and Footers Document should have running headers and footers (Experimental: Content of headers and footers vary widely, are not always predictable and should be manually verified). EXPLANATION Fail	Submit Buttons Forms should contain proper submit buttons to allow submission by keyboard. EXPLANATION N/A
Natural Language Document should contain a specification of the natural language. EXPLANATION Pass	Page Numbering Document pages should be numbered consistently to ease navigation to referenced pages. EXPLANATION N/A	Document Title Document should have a document title. EXPLANATION Fail	Link Text for External Links Link text should convey its purpose. EXPLANATION N/A

Рисунок 1.9 - Звіт про доступність наданий Tingtun Checker

РАС

Історія розвитку РАС, безкоштовного аналізатора доступності PDF-документів, демонструє його постійне вдосконалення. У 2010 році з'явився РАС 1 Beta з унікальною можливістю візуалізації роботи зчитувачів екрану (див. Рисунок 1.10). У 2012-му РАС 1.3 додав функцію детального звіту (див. Рисунок 1.11), а в 2014-му РАС 2 став першим інструментом для перевірки відповідності стандарту PDF/UA-1. У 2017 році РАС 3 забезпечив створення спрощених текстових звітів, а РАС 2021, випущений у 2021 році, отримав підтримку стандарту WCAG 2.0. У листопаді 2023 року РАС отримав власний сайт із підтримкою axes4, а РАС 2024 розширив можливості перевірки якості для ручного тестування доступності [30].

Не зважаючи на те що РАС не має програмного інтерфейсу або можливостей інтеграції axes4 також надає можливість генерацію спрощеного звіту онлайн за допомогою інструменту axesCheck [31].

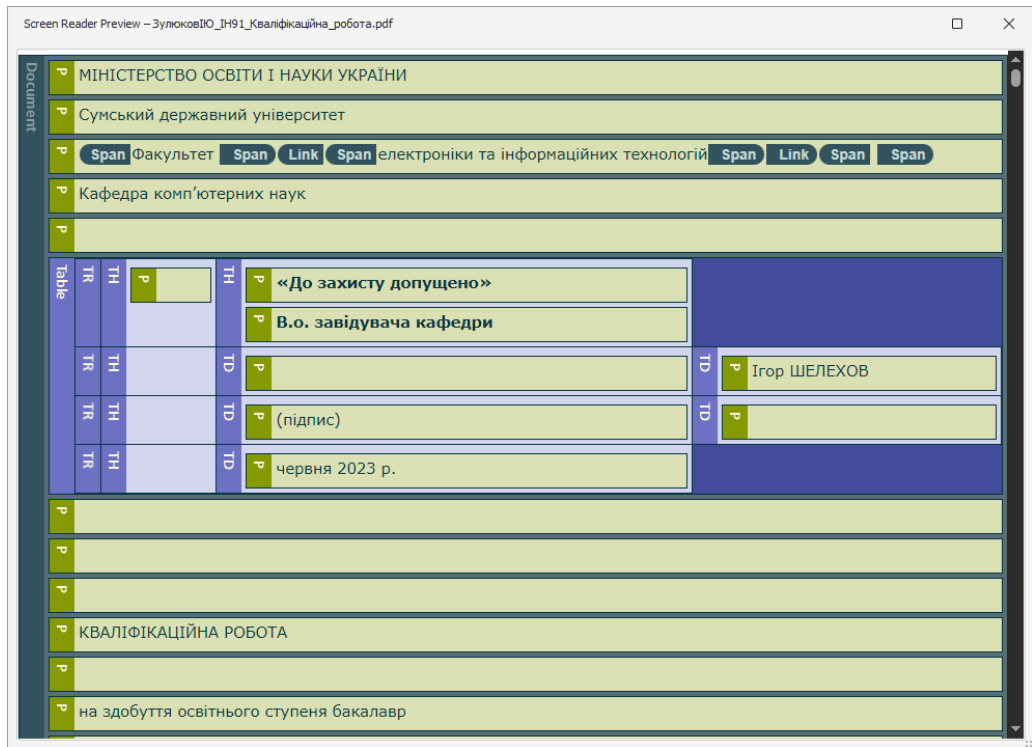


Рисунок 1.10 - Візуалізація роботи зчитувачів з екрану

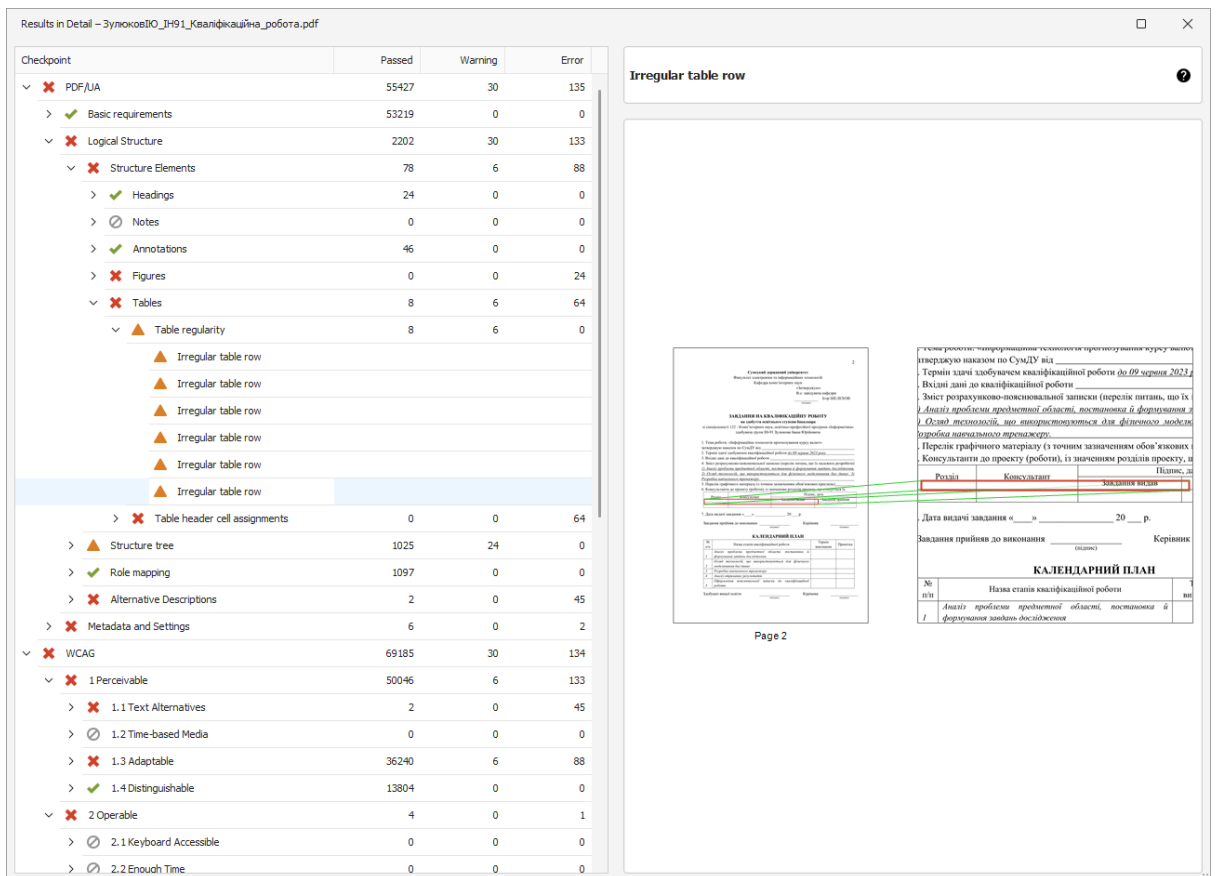


Рисунок 1.11 - Детальний звіт згенерований PAC

Переваги:

- Безкоштовний інструмент
- Візуалізація роботи зчитувачів з екрану
- Підтримка PDF/UA-1, PDF/UA-2 та WCAG 2.0

Недоліки:

- Спрощений звіт недостатньо інформативний
- Відсутність можливості інтеграції у власний додаток

PDF/UA Foundation Grackle

Через те що більшість публікацій рекомендує PAC було написано до того як PAC отримав власний сайт велика кількість користувачів опиниться на сайті PDF/UA Foundation де їх зустріне реклама інструменту Grackle. Оскільки цей інструмент має майже ту ж саму мету що і PAC дуже вірогідно що більшість користувачів вирішить що Grackle це нова версія PAC але Grackle більш схожий на CommonLook. Пакет інструментів включає [32]:

- Grackle Workspace – плагін для Google Docs, Sheets та Slides
- Grackle PDF – інструмент для валідації та редагування PDF файлів
- Grackle GO – безкоштовний онлайн PDF валідатор
- Grackle Stream – інструмент для автоматизації PDF валідації
- Grackle Scan та Scan for WP(WordPress) – інструменти для перевірки усіх PDF на сайті

Переваги:

- Можливість інтеграції в Google Workspace
- Можливість автоматизації PDF валідації

Недоліки:

- Відсутність інтуїтивного шляху придбати або завантажити продукт

VeraPDF

VeraPDF охоплює всі версії і рівні відповідності PDF/A та обидві версії PDF/UA. Подібно pdfaPilot цей інструмент доступний для використання на робочому столі [33] та готовий для інтеграції в існуючі системи за допомогою інтерфейсу командного рядка [34] чи набору програмного забезпечення [35]. На

відміну від pdfaPilot, veraPDF має повністю відкритий код та є абсолютно безкоштовним. Це дозволяє розробникам більш тісно ознайомитись з влаштуванням інструменту та краще інтегрувати його у власний додаток. Більш того це дозволяє розробникам поступово вносити правки в код veraPDF що постійно робить цей інструмент найсучаснішим та забезпечує повну відповідність стандартам PDF/A та PDF/UA.

Переваги:

- Можливість автоматизації
- Безкоштовний інструмент
- Наявність відкритого коду
- Повна підтримка стандартів PDF/A та PDF/UA

Недоліки:

- Інструмент розрахований для використання експертами через що звіт занадто технічний та абсолютно незрозумілий для пересічного користувача(див. Рисунок 1.12)

Rule	Status
Specification: ISO 32005:2023, Clause: 6.2, Test number: 275 <TOCI> shall not contain content items 1 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.11.1, Test number: 1 The Metadata stream as specified in ISO 32000-2:2020, 14.3 in the document catalog dictionary shall contain a dc:title entry 1 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.8, Test number: 1 All destinations whose target lies within the current document shall be structure destinations 44 occurrences	Failed Show
Specification: ISO 32005:2023, Clause: 6.2, Test number: 247 <TOCI> shall not contain 22 occurrences	Failed Show
Specification: ISO 32005:2023, Clause: 6.2, Test number: 249 <TOCI> shall not contain <Link> 22 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.2.5.2, Test number: 2 The structure tree root shall contain a single Document structure element as its only child. The namespace for that element shall be specified as the PDF 2.0 namespace 1 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 5, Test number: 1 The PDF/UA version of a file shall be specified in the value of the Metadata entry in the document catalog using the PDF/UA identification schema 1 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.2.5.28.2, Test number: 1 A Figure structure element shall have at least one of the following properties: a) an alternate description (Alt property), as specified in ISO 32000-2:2020, 14.9.3; b) a replacement text (ActualText property) that represents the content enclosed by the Figure structure element 22 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.2.5.8, Test number: 1 Each TOCI in the table of contents shall identify the target of the reference using the Ref entry, either directly on the TOCI structure element itself or on one of its child structure elements 23 occurrences	Failed Show
Specification: ISO 14289-2:2024, Clause: 8.2.5.26, Test number: 3 Tables shall be regular. Table rows shall have the same number of columns (taking into account column spans) 3 occurrences	Failed Show

Рисунок 1.12 – Приклад HTML звіту VeraPDF щодо відповідності PDF/UA-2

Підсумок

Таблиця 1.1 Порівняння інструментів оцінки доступності документів

	Ціна	Версія WCAG	Версія PDF/UA	Автоматизація	Веб додаток
Adobe Accessibility Checker	30\$/ місяць	2.0	1	закритий доступ	відсутній
allyant CommonLook	не вказана	не вказана	не вказана	закритий доступ	закритий доступ
Adobe Accessibility Auto-Tag	30\$/ місяць	-	-	500 запитів/ місяць	500 запитів/ місяць
PAVE	безкоштовно	-	-	відсутня	присутній
callas software pdfaPilot	від 625€ до 6625€	-	не вказана	CLI, Server	відсутній
ЕПІ Tingtun Checker	безкоштовно	2.0	-	CLI	ASCHECKS
PAC	безкоштовно	2.0	1 та 2	відсутня	відсутній
PDF/UA Foundation Grackle	не вказана	не вказана	не вказана	закритий доступ	закритий доступ

	Ціна	Версія WCAG	Версія PDF/UA	Автоматизація	Веб додаток
VeraPDF	безкоштовно	2.0	1 та 2	CLI, Java бібліотека, API	REST демо та PDF4WCA G

В результаті огляду існуючих інструментів(див. Таблиця 1.1) було вирішено що головними проблемами є легкодоступність інструментів та зрозумілість звітів. Під легкодоступністю мається на увазі легкість отримання, налаштування та використання інструменту. Значна кількість інструментів потребують спілкування з представником та заключення контракту на використання інструментів що значно обмежує доступ до інструменту для малих або некомерційних організацій. CLI інструменти не потребують встановлення але потребують навиків користування командним рядком. GUI інструменти потребують встановлення та налаштування системним адміністратором. Найбільш легкодоступними є Web інструменти але їх кількість доволі обмежена.

Зрозумілість звіту залежить як від інструменту так і від стандарту якому повинен відповідати документ. Звіти CLI інструментів занадто технічні та дуже важкі для сприйняття навіть для експертів з доступності. Звіти GUI інструментів зазвичай поділяються на спрощені та детальні але єдиною перевагою детальних звітів є спрощення місцезнаходження проблеми в документі. Розробники інструментів не вважають потрібним та зазвичай не мають можливості пояснити важливість кожної проблеми та як їх виправити. У випадку WCAG розробники мають можливість надати посилання на офіційні пояснення та шляхи виправлення і все одно не завжди користуються цією можливістю. З іншого боку для WTPDF, PDF/A, PDF/UA та інших стандартів користувач повинен сам знайти стандарт та самостійно знайти в стандарті проблему та її пояснення. При цьому якщо WTPDF є безкоштовним то інші стандарти доволі дорого коштують що

знову ж таки обмежує легкодоступність. Більш того Web хоч і найбільш легкодоступні але зазвичай надають лише спрощені звіти що можуть не містити окремі проблеми взагалі а відображати лише загальну оцінку доступності документу.

Таким чином заохочується нехтування доступністю офісних документів з боку малих організацій. виправити цю проблему можливо за допомогою штучного інтелекту але доволі мала кількість моделей генерації тексту дозволяють завантаження файлів. Більш того хоча штучний інтелект і має загальне розуміння цієї сфери, він не здатен на більш точну оцінку доступності. Обидві з цих проблем можна вирішити об'єднанням точності існуючого інструменту аналізу документів з семантичними можливостями штучного інтелекту. Огляд показав що подібного рішення не існує що спонукало на створення власного додатку.

1.3 Постановка задачі

Мета роботи: розробка інтелектуальної технології оцінки доступності документів у форматі PDF.

Розроблюваний додаток повинен:

1. Бути безкоштовним та не потребувати завантаження та встановлення
2. Підтримувати якомога більше стандартів доступності, зокрема новітні стандарти PDF/UA-2 та WCAG 2.2
3. Застосовувати ряд аналізаторів, специфічних до формату документу
4. Забезпечувати агрегацію та гармонізацію отриманих звітів із використанням ряду великих мовних моделей

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Вибір мови програмування

Оскільки метою роботи є створення легкодоступного додатку, було вирішено обрати саме веб-додаток, адже він має низку переваг у порівнянні з іншими типами додатків. По-перше, веб-додаток не потребує встановлення на пристрої користувача — він доступний через будь-який браузер. Це спрощує доступ до нього з будь-якого пристрою, що має доступ до Інтернету, і позбавляє необхідності турбуватися про сумісність з різними операційними системами.

По-друге, веб-додаток дозволяє зручніше здійснювати оновлення та підтримку, оскільки всі зміни відбуваються на серверній частині, і користувачі автоматично отримують доступ до актуальної версії програми без додаткових дій зі свого боку. Це особливо важливо для додатків, які планується регулярно вдосконалювати чи додавати новий функціонал.

Крім того, веб-додаток може забезпечити кращу інтеграцію з іншими сервісами та ресурсами в Інтернеті, а також легше адаптуватися під потреби користувачів, наприклад, шляхом налаштування під різні пристрої через адаптивний дизайн.

Виходячи з цього, було обрано мову програмування JavaScript, оскільки вона є основною мовою для створення інтерактивних веб-додатків. JavaScript має низку переваг, які роблять її оптимальним вибором для розробки веб-застосунків:

1. JavaScript підтримується усіма сучасними браузерами і є мовою, яка виконується на боці клієнта (front-end), що дозволяє створювати динамічні та інтерактивні інтерфейси без необхідності перезавантаження сторінки. Це покращує зручність та швидкість роботи користувачів із додатком.

2. JavaScript має потужну екосистему бібліотек та фреймворків, таких як React, Angular, Vue.js, що спрощують процес розробки та дозволяють створювати додатки будь-якої складності. Крім того, ці інструменти

забезпечують додаткові можливості для швидкої розробки, тестування та підтримки коду.

3. Завдяки Node.js, JavaScript можна використовувати не лише на клієнтській, а й на серверній частині додатка (back-end). Це дозволяє створити єдине середовище для всього проєкту, що спрощує розробку, підтримку та взаємодію між клієнтською і серверною частинами додатка.

4. JavaScript є однією з найпопулярніших мов програмування, що забезпечує доступ до великої кількості навчальних ресурсів, документації та інструментів. Велика спільнота розробників також полегшує вирішення технічних проблем і забезпечує постійний розвиток та покращення мови і пов'язаних з нею технологій.

З огляду на всі ці переваги, вибір JavaScript для розробки веб-дodatка виглядає обґрунтованим і оптимальним, враховуючи вимоги до доступності, інтерактивності та підтримки додатка.

2.2 Вибір API

Для досягнення мети було поставлено задачі використання інструментів що виходять за межі бібліотек: валідатора документів та штучного інтелекту. В якості валідатора було обрано VeraPDF так як інші рішення або не мають можливості інтеграції в інші додатки або не є безкоштовними. VeraPDF також має вичерпну документацію та є визнаним індустрією інструментом що робить його ідеальним вибором для цієї задачі. Для ролі штучного інтелекту було обрано платформу Hugging Face через великий вибір безкоштовних моделей створених гігантами індустрії такими як OpenAI, Meta, Google та інші.

Для забезпечення легкодоступності потрібно максимально зменшити витрати на хостинг додатку. Для досягнення цієї мети потрібно винести важкі та енергозатратні обчислення за межі додатку. Це було зроблено за допомогою надсилання запитів до програмних інтерфейсів вже розгорнутих додатків та отримання готових результатів.

VeraPDF REST API

Даний додаток є простим REST сервісом що реалізує можливості VeraPDF. Додаток також має простий веб інтерфейс розгорнутий за адресою demo.verapdf.org. Підтримуються лише вбудовані в VeraPDF профілі з яких в даній роботі використовуються тільки PDF/A Рівня А через їх недоступність у наступному додатку.

В даній роботі було використано лише розташування `/api/validate/{profileId}` цього додатку. Додаток отримує PDF файл, перевіряє його за обраним профілем та надсилає результат у одному з форматів: HTML(за замовчуванням), JSON та XML. В даній роботі обрано JSON для уніфікації результатів та полегшення обробки враховуючи використання мови JavaScript.

VeraPDF Web Server API

Даний додаток є серверною частиною інструменту pdf4wcag.com. Додаток також реалізує можливості VeraPDF але на відміну від попереднього реалізує керування файловими ресурсами, робітниками та задачами через що є набагато складнішим у використанні. Більш того з вбудованих профілів підтримуються лише обидві версії PDF/UA. Додатково підтримуються експериментальні профілі такі як WCAG 2.2, WTPDF 1.0 та інші. Також можливе опитування серверу під час виконання аналізу для отримання індикації прогресу.

В даній роботі використовуються розташування `/validate/api/files` для завантаження PDF документу та JSON результату і `/validate/api/jobs` для створення та запуску робітника з задачею, та отримання статусу виконання задачі.

Hugging Face Serverless Inference API

Платформа Hugging Face займається публікацією різноманітних моделей штучного інтелекту. Ці моделі можна завантажити для локального використання, розгорнути за допомогою Inference Endpoints або використати вже розгорнуті за допомогою Serverless Inference.

В даній роботі було використано саме Serverless Inference API так як вибір вже розгорнутих моделей більш ніж достатній для цієї роботи. Окрім широкого

вибору моделей Hugging Face Serverless Inference API також надає найкращий безкоштовний тариф: 1000 запитів на день. Цей тариф має не тільки найбільшу кількість та найшвидше оновлення але і найменш строгий критерій рахування адже запит може бути будь-якого розміру та бути надісланий до моделі будь-якого розміру тоді як інші сервіси рахують тарифи в токенах або обчислювальній вартості.

2.3 Вибір бібліотек

Вибір бібліотек для цього проєкту обґрунтований необхідністю забезпечити зручну розробку, кросбраузерну підтримку, а також покращення взаємодії користувача з інтерфейсом.

Meta React

Основна бібліотека для побудови користувацьких інтерфейсів, яка забезпечує компонентний підхід до розробки. Вона дозволяє створювати незалежні компоненти, що управляють власним станом і автоматично оновлюються у відповідь на зміну даних, що спрощує розробку інтерактивних веб-додатків.

Meta React-DOM

Використовується для інтеграції компонентів React із браузером, оскільки вона рендерить компоненти безпосередньо у DOM структуру веб-сторінки. Це необхідно для забезпечення коректного відображення додатка в браузері.

Material-UI Material

Надає набір компонентів для побудови сучасного інтерфейсу користувача відповідно до принципів Material Design. Це забезпечує користувачам знайомий і зрозумілий інтерфейс.

Material-UI Icons

Набір іконок, який використовується разом з компонентами Material-UI для додавання візуальних елементів до інтерфейсу. Іконки підвищують зрозумілість і зручність використання інтерфейсу.

Emotion React

Бібліотека для гнучкої стилізації компонентів, яка використовується у Material-UI для додавання стилів до компонентів. Emotion React підтримує динамічні стилі, що дозволяє адаптувати інтерфейс до стану компонентів та інших умов.

Emotion Styled

Додатковий пакет для стилізації, який дозволяє створювати стилізовані компоненти за допомогою шаблонних рядків, роблячи код більш читабельним та полегшуючи підтримку кастомних стилів.

Hugging Face Inference

Бібліотека для інтеграції з моделями штучного інтелекту через API Hugging Face. Це дозволяє додатку використовувати потужні можливості обробки природної мови для синтезу пояснень звітів доступності та шляхів виправлення наявних проблем.

React Markdown

Забезпечує можливість відображення контенту у форматі Markdown безпосередньо у додатку. Це дозволяє зберегти форматування Markdown яке доволі часто використовується у відповідях штучного інтелекту.

Spark MD5

Дана бібліотека використовується для генерації MD5 хешу що потрібен при надсиланні PDF файлу до VeraPDF Web Server API. Використання цієї бібліотеки максимально прискорює генерацію хешу порівняно з іншими шляхами завдяки використанню найшвидшого алгоритму та додаткових оптимізацій.

2.4 Додаткові інструменти

Create React App, Vite, і Next.js — популярні інструменти для створення та розробки сучасних веб-додатків на базі JavaScript. Кожен із них має свої особливості та підходить для різних завдань.

Create React App

Офіційний інструмент від команди React для швидкого створення проєктів на основі React. Його головні особливості:

- генерує готовий проєкт з налаштованими залежностями, що дозволяє розпочати розробку без конфігурації Webpack або Babel.
- надає оптимізовану для продакшн збірку, яка покращує продуктивність і зменшує розмір файлів.
- приховує більшість конфігурацій від розробника, що зручно для початківців, але іноді обмежує можливості кастомізації.

Create React App підходить для створення базових React-додатків, але має певні обмеження щодо швидкості збірки у великих проєктах, а також обмежену підтримку SSR (серверного рендерингу).

Vite

Сучасний інструмент для розробки, орієнтований на швидкість і оптимізацію процесу збірки. Основні переваги:

- використовує сучасні можливості браузерів, такі як ES-модулі, що зменшує час на початкове збирання і перезавантаження сторінок.
- забезпечує дуже швидке оновлення модулів без перезавантаження всієї сторінки, що значно пришвидшує розробку.
- пропонує простий і зручний механізм налаштування за допомогою `vite.config.js`, дозволяючи легко змінювати параметри проєкту та інтегрувати додаткові плагіни.

Next.js

Фреймворк для React, який надає розширені можливості для створення веб-додатків:

- дозволяє рендерити сторінки на сервері, що підвищує швидкість завантаження сторінки та оптимізує SEO.
- дозволяє створювати статичні сторінки під час білду, що підходить для інформаційних сайтів і блогів.

- включає вбудований маршрутизатор, який дозволяє легко налаштовувати маршрути за допомогою файлової структури.
- дає можливість обробки серверних запитів без необхідності додаткових серверних фреймворків.

Next.js підходить для розробки більш складних додатків з потребою в SSR або SSG. Однак для SPA (односторінкових додатків), де основна увага приділяється швидкості розробки та продуктивності клієнтської частини, його можливості можуть бути надлишковими.

Обґрунтування вибору

Враховуючи мету створення легкодоступного, швидкого та зручного у використанні веб-додатку, формат SPA є оптимальним рішенням. SPA забезпечує швидкість, інтерактивність і легкість управління інтерфейсом, що є важливими аспектами для сучасних додатків. SPA також дозволяє швидко та ефективно надавати користувачам доступ до основного функціоналу та оновлювати контент у реальному часі.

Для SPA, де основне навантаження припадає на клієнтську частину, Vite є оптимальним вибором. Він не має надмірної функціональності для серверного рендерингу, як у Next.js, що робить його більш ефективним для даного типу проєкту.

2.5 Проєктування інтелектуальної технології

Проєктування є важливим етапом розробки інтелектуальної технології, оскільки дозволяє створити чітку архітектуру системи, забезпечити узгодженість між її компонентами та визначити взаємодію між ними. Це спрощує подальшу розробку, тестування та впровадження технології. Для цього було створено ряд діаграм.

Діаграма прецедентів

На першому етапі проєктування важливо визначити основні функції системи та взаємодію користувачів з її компонентами. Для цього

використовуються діаграми прецедентів (Use Case Diagram), які дозволяють візуалізувати ключові сценарії взаємодії з системою.

Діаграма прецедентів є одним із інструментів UML (Unified Modeling Language) і використовується для опису функціональності системи з точки зору користувачів. Основними елементами діаграми є:

- Актори – зовнішні об'єкти (люди, системи або пристрої), що взаємодіють із системою.
- Прецеденти – дії, які система виконує для досягнення конкретної мети актора.
- Зв'язки – лінії, які показують взаємодію між акторами та прецедентами.

У створеній діаграмі (див. Рисунок 2.1) головними прецедентами є аналіз доступності документа та уточнення відповіді штучного інтелекту. Акторами є:

- Користувач, що ініціює головні процеси.
- API аналізатору доступності документів VeraPDF.
- API штучного інтелекту Hugging Face Serverless Inference.

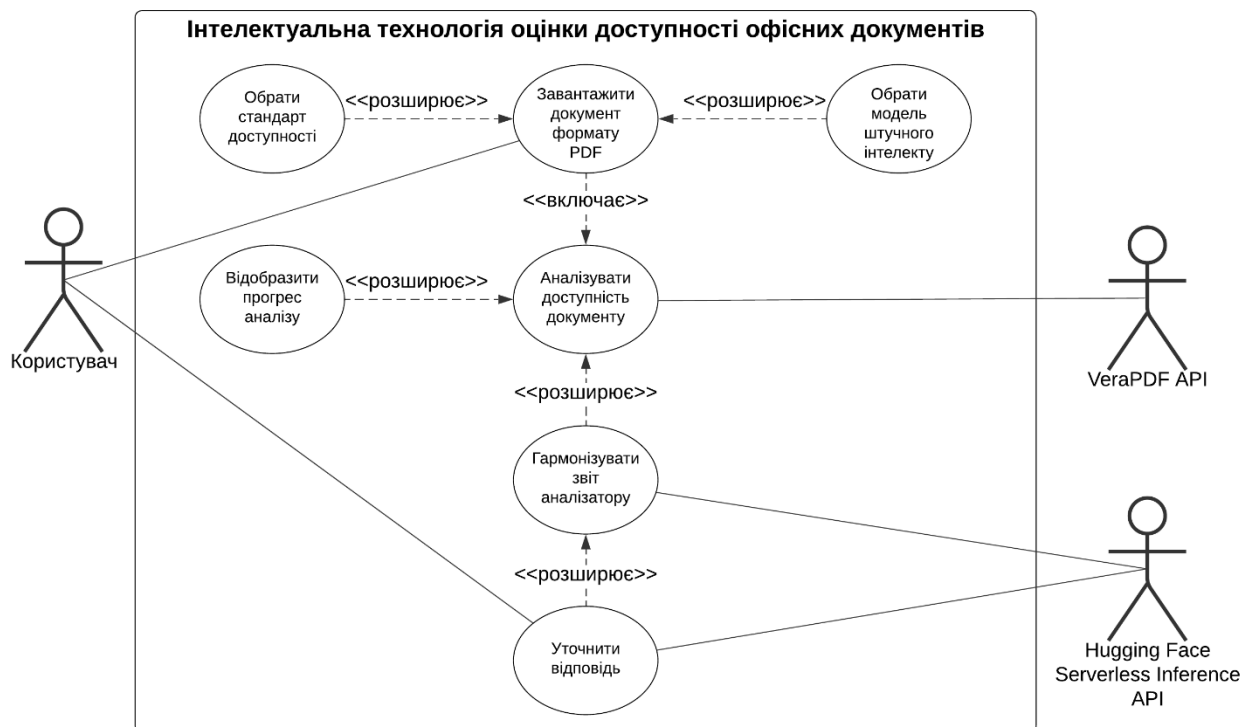


Рисунок 2.1 - Діаграма прецедентів інтелектуальної технології

Таким чином, за допомогою діаграми прецедентів вдалося визначити ключові сценарії взаємодії між користувачами та системою, що дозволило створити чітку структуру функціональних вимог для подальшої реалізації технології.

Опис функціональних процесів додатку у нотації IDEF0

Після визначення основних функцій системи, наступним етапом є більш детальне моделювання функціональних процесів, їхніх вхідних і вихідних даних, а також ресурсів, необхідних для їх виконання. Це дозволяє побудувати структуровану картину процесів і забезпечити оптимальну взаємодію між компонентами. Для цього використовуються діаграми нотації IDEF0, які допомагають візуалізувати та формалізувати ці процеси.

IDEF0 – це методологія графічного моделювання, яка використовується для опису функціональних процесів системи. Основними елементами нотації є:

- Функціональні блоки – прямокутники, які представляють дії або функції системи.
- Стрілки – використовуються для позначення входів (вхідних даних), виходів (результатів), механізмів (ресурсів) і обмежень, що впливають на виконання функції.

Створено IDEF0 діаграму для двох ключових функцій додатку: аналізу доступності документу з подальшою гармонізацією звіту(див. Рисунок 2.2) та уточнення відповіді штучного інтелекту(див. Рисунок 2.3).

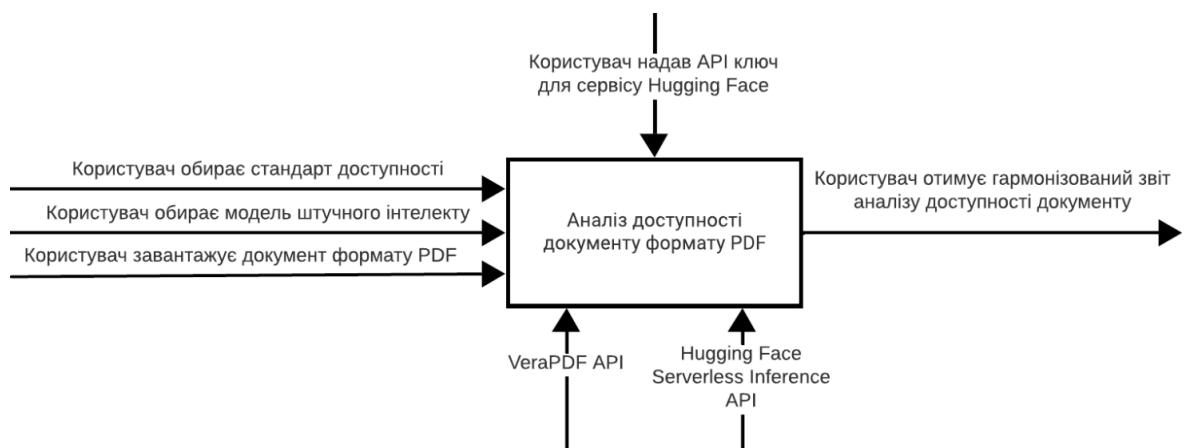


Рисунок 2.2 - IDEF0 діаграма функціоналу «Аналіз доступності документу формату PDF»

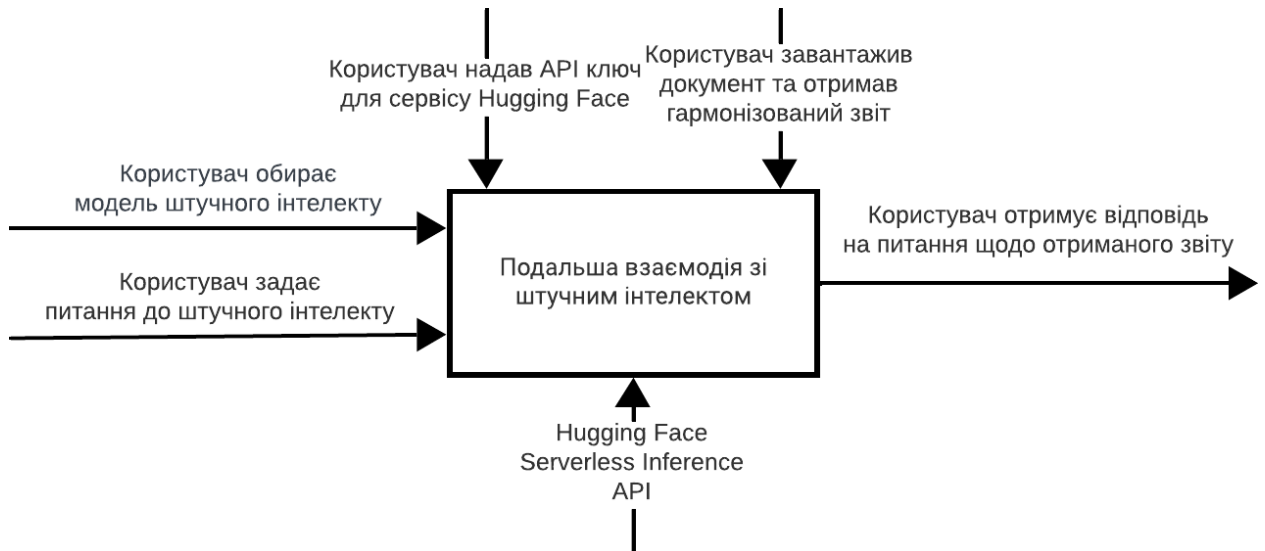


Рисунок 2.3 - IDEF0 діаграма функціоналу «Подальша взаємодія зі штучним інтелектом»

Як результат, детально описавши функціональні процеси за допомогою нотації IDEF0, ми змогли чітко візуалізувати всі етапи роботи системи та ресурси, необхідні для їх виконання, що допоможе ефективно організувати роботу над додатком.

Діаграми послідовності

Завершальним етапом проектування є деталізація взаємодії між компонентами системи з урахуванням порядку виконання операцій. Для цього використовуються діаграми послідовності, які дозволяють візуалізувати послідовність дій і кроків, необхідних для виконання функцій.

Діаграма послідовності – це один із видів UML-діаграм, що описує порядок взаємодії між об'єктами у системі. Основними елементами є:

- Об'єкти – учасники процесу, представлені у вигляді прямокутників.
- Лінії життя – вертикальні лінії, що позначають часову шкалу існування об'єкта в рамках процесу.
- Повідомлення – стрілки, які описують передачу інформації або виклики функцій між об'єктами.

Створену діаграму послідовності було розділено на дві частини: послідовність використання VeraPDF REST API(див. Рисунок 2.4) та

послідовність використання більш складного VeraPDF Web Server API(див. Рисунок 2.5).

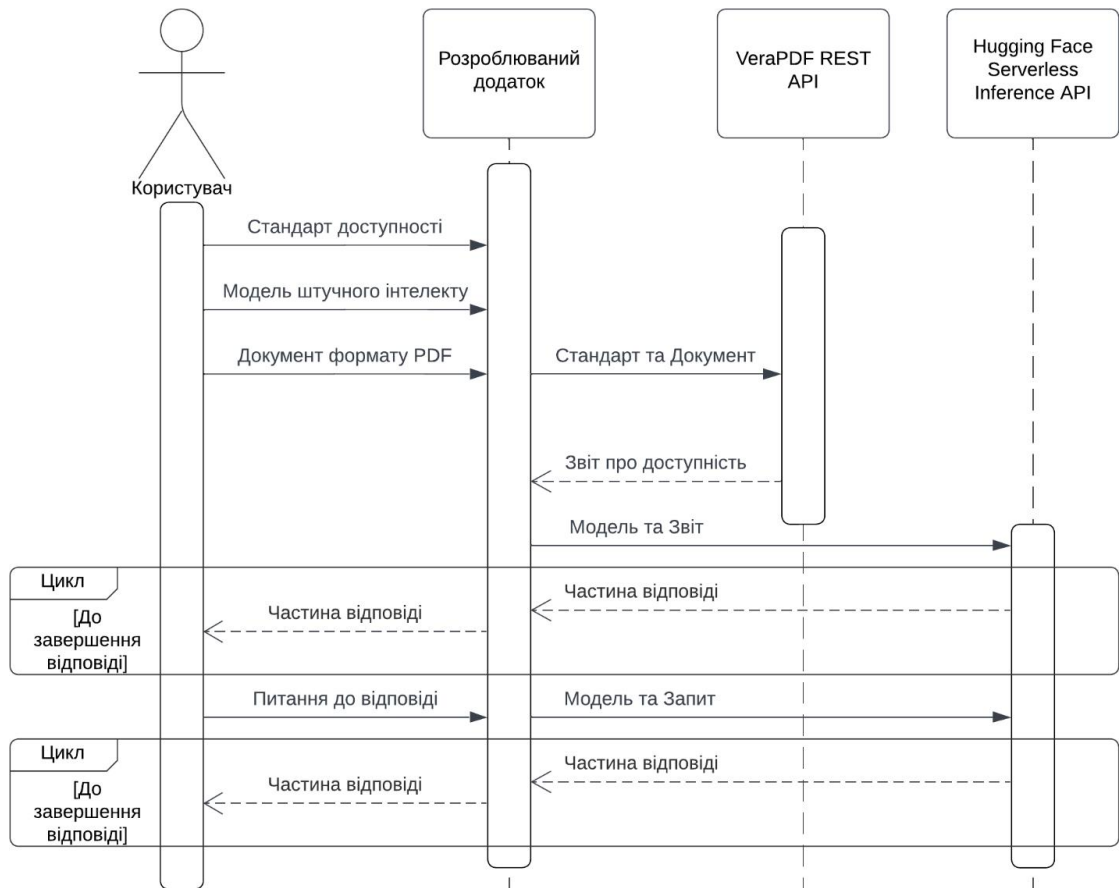


Рисунок 2.4 - Діаграма послідовності для додатку при умові використання VeraPDF REST API

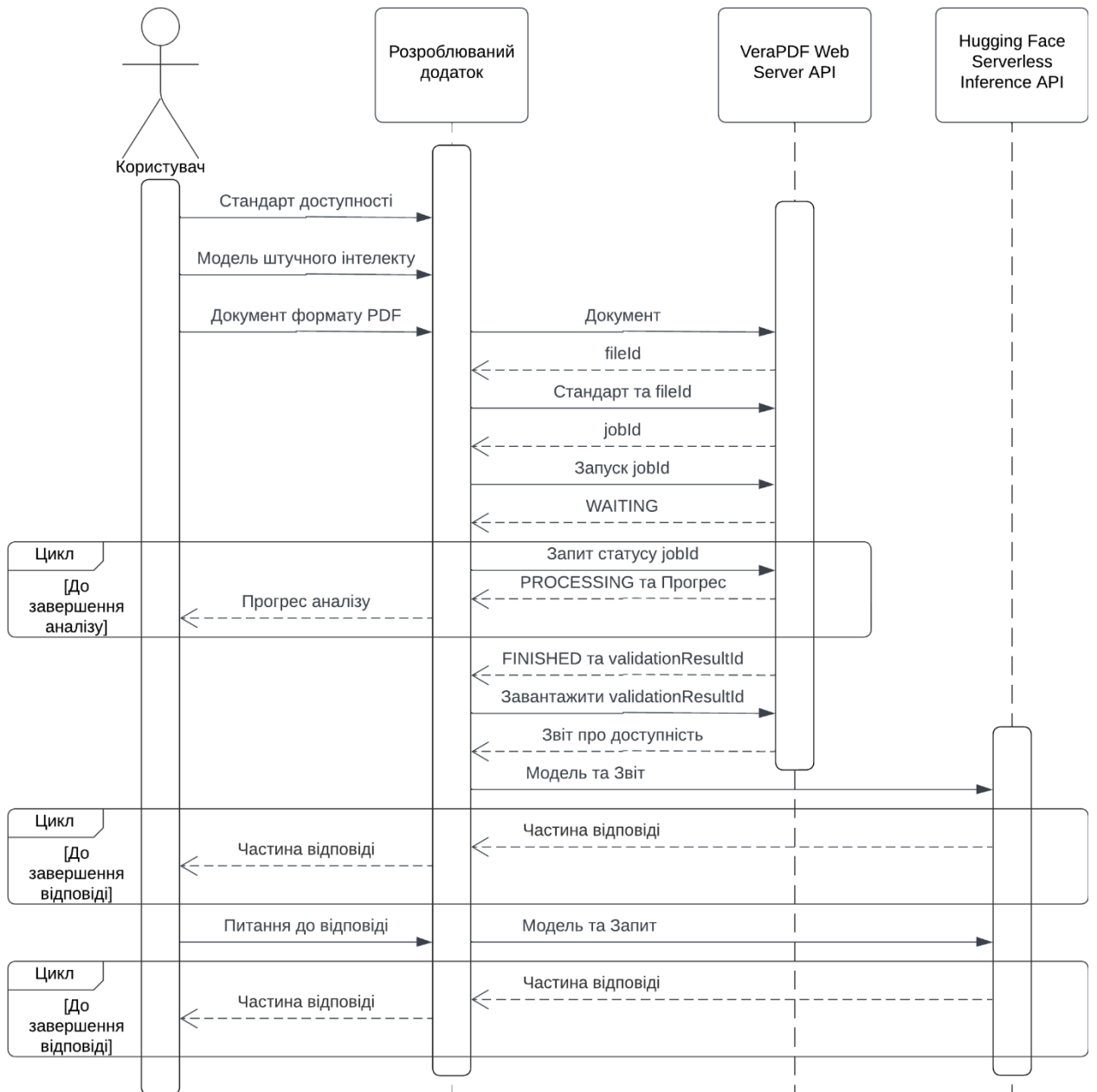


Рисунок 2.5 - Діаграма послідовності для додатку при умові використання VeraPDF Web Server API

Таким чином, побудувавши діаграми послідовності, ми змогли точно визначити порядок виконання основних операцій та взаємодії між компонентами системи, що дозволить оптимізувати процеси та зменшити ймовірність помилок у роботі додатку.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Формування вхідних даних

В якості документу що не відповідає жодному зі стандартів доступності було обрано власну кваліфікаційну роботу бакалавра ЗулюковІЮ_ІН91_Кваліфікаційна_робота.pdf.

Тестові файли для граничного випадку коли документ відповідає вимогам було взято з репозиторію проєкту LaTeX. Для PDF/UA-1 було обрано macbeth-tagged-17.pdf. Для PDF/UA-2 було обрано macbeth-tagged.pdf.

3.2 Опис програмної реалізації

Оскільки додаток було створено у форматі SPA більшість коду знаходиться у файлі App.jsx. Даний файл є React компонентом що завантажується за допомогою скрипту main.jsx після завантаження index.html.

Всередині файлу визначена функція App що повертає інтерфейс додатку в MUI компоненті Container. При виконанні функції App виконується функція useEffect що відображає діалогове вікно(див. Рисунок 3.1) що питає у користувача API ключ для Hugging Face а також надає посилання на офіційний посібник для безкоштовного отримання цього ключа.

Please enter a valid Hugging Face API key

.....|

[Get a token](#)

VALIDATE

Рисунок 3.1 - Діалогове вікно для вводу API ключа

Такий підхід перекладає відповідальність за обмежений доступ до Hugging Face Inference API на користувача що дозволяє зменшити витрати на хостинг

додатку та забезпечити його безкоштовність. З іншого боку це значно збільшує кількість запитів що може виконати кожен користувач.

При натисненні кнопки `VALIDATE` викликається функція `handleValidate` що перевіряє відповідність ключа загальному формату API ключів Hugging Face: `^hf_[a-zA-Z]{34}$`. Ключ не буде надісланий якщо користувач зробив ввів довільний текст замість отримання ключа згідно наведеного посібника. Це дозволяє зменшити навантаження на сервера Hugging Face та запобігти блокуванню додатку. Успішно введений ключ зберігається в пам'яті вкладки браузера що дозволяє не вводити ключ для кожного запиту або відкривати декілька вкладок для декількох ключів. Після закриття вкладки ключ видаляється з пам'яті браузера.

Далі користувач повинен обрати стандарт відповідності якому буде перевіряти додаток(див. Рисунок 3.2). Для цього створено `FormComponent` всередині якого знаходиться підпис `InputLabel` та випадаючий список `Select`.

Користувачу доступно на вибір дев'ять стандартів:

- PDF/A-1A
- PDF/A-2A
- PDF/A-3A
- PDF/UA-1
- PDF/UA-2
- PDF/UA-2 & ISO 32005
- WTPDF 1.0 Accessibility
- WCAG 2.2 Machine
- WCAG 2.2 Machine & Human

Застарілі стандарти PDF/A наведені на випадок якщо відповідність будь-якому з цих стандартів потребує організація користувача. Базові та інші рівні цих стандартів відсутні тому що мета додатку – перевірка саме на доступність. На відміну від інших ці стандарти не будуть відображати процес аналізу але

враховуючи високу швидкість перевірки цим можна знехтувати. За замовчуванням обраний найновітніший стандарт – PDF/UA-2 & ISO 32005.

Після цього користувач повинен обрати модель штучного інтелекту що буде генерувати пояснення та рішення проблем знайдених валідатором.

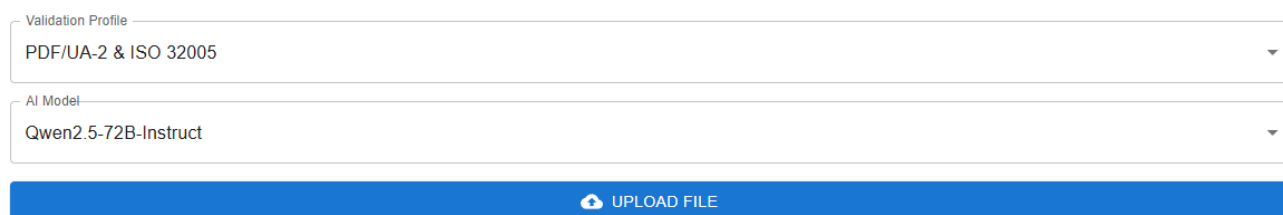
За доступністю через Serverless Inference API моделі поділяються на:

- Warm – готові до використання
- Cold – не готові але доступні для використання
- Frozen – не доступні для використання

В даній роботі був використаний курований список Warm моделей:

- Qwen2.5-72B-Instruct створена компанією Alibaba
- Llama-3.1-70B-Instruct створена компанією Meta
- Mixtral-8x7B-Instruct-v0.1 створена компанією Mistral AI
- gemma-2-9b-it створена компанією Google

Огляд типів моделей показав що більше всього для поставленої задачі підходять моделі текстової генерації налаштовані на дотримання інструкцій або іншими словами чатові моделі. Додатковим критерієм вибору враховуючи очікуваний розмір запиту став розмір контексту. Ціллю було поставлено 32 тисячі токенів(слів). Gemma підтримує лише 8 тисяч але може самостійно розширити підтримку до 32 тисяч. Подібно до цього Qwen може розширити підтримку з 32 до 128 тисяч токенів. За замовчуванням обрана найбільша модель – Qwen2.5-72B-Instruct.



The image shows a user interface for selecting a validation profile and an AI model. It features two dropdown menus. The first dropdown, labeled 'Validation Profile', has 'PDF/UA-2 & ISO 32005' selected. The second dropdown, labeled 'AI Model', has 'Qwen2.5-72B-Instruct' selected. Below these dropdowns is a prominent blue button with the text 'UPLOAD FILE' and a small cloud icon to its left.

Рисунок 3.2 - Головний інтерфейс додатку

Після того як користувач обрав стандарт та модель він повинен натиснути кнопку UPLOAD FILE. Ця кнопка відкриє діалогове вікно з файловою системою

користувача де користувач може обрати будь-який файл PDF. Користувач обмежений вибором лише файлів формату PDF.

Після вибору файлу користувач натискає на кнопку Відкрити що завантажує файл та надсилає його до валідатора. Для уникнення проблем з CORS використовується проксі сервер визначений в конфігураційному файлі vite.config.js. Завантаження файлу виконується функцією handleFileChange. Якщо був обраний один з трьох профілів PDF/A файл надсилається функцією validatePdf файлу apiHandler.js що повертає відповідь API у форматі JSON. Якщо ж був обраний будь-який інший профіль файл надсилається функцією uploadFile що додатково обчислює MD5 хеш файлу та повертає відповідь файлового сервісу що містить ID збереженого файлу. Отримавши ID файлу виконується функція createJob що створює робітника та задачу на основі файлу і повертає ID робітника. Виконання задачі запускається функцією executeJob. Після цього у циклі do while кожену секунду додаток опитує сервер про статус виконання задачі за допомогою функції checkJob що повертає відповідь серверу зі статусом та прогресом валідації. Коли виконання успішно завершується ця функція також повертає ID файлу з результатом у форматі JSON. Вміст цього файлу отримується за допомогою функції getValidationResult.

Отриманий JSON оброблюється функцією parseJson файлу jsonHandler.js. З результату вибирається лише найпотрібніша інформація для запобігання перевищення ліміту токенів моделі штучного інтелекту.

Вибрана інформація додається до контексту та надсилається в текстовому форматі до Hugging Face Inference API за допомогою функції getTextGenerationStream файлу aiHandler.js. Дана функція замість однієї відповіді повертає потік частин відповіді що дозволяє отримати результат поступово та запобігає довгому очікуванню відповіді серверу.

Для відображення текстової інформації на екрані використовується динамічний масив статичних текстових контейнерів ReactMarkdown та Typography і один динамічний текстовий контейнер ReactMarkdown.

Динамічний контейнер дозволяє реалізувати відображення отриманого на попередньому кроці потоку частин відповіді штучного інтелекту. Поступове відображення запобігає очікуванню реакції від додатку та помилкового повторного надсилання файлу. Окрім цього цей текстовий контейнер використовується для відображення інструкції з використання додатку та прогресу аналізу файлу.

Використання ReactMarkdown дозволяє повністю зберегти будь-яке форматування що покращує зрозумілість та структурованість відповіді штучного інтелекту.

Динамічний масив використовується для відображення додаткової взаємодії користувача з штучним інтелектом. Після отримання всіх частин відповіді динамічний ReactMarkdown містить повну відповідь штучного інтелекту та користувачеві надається можливість задати штучному інтелекту питання за допомогою текстового поля та кнопки Send (див. Рисунок 3.3).

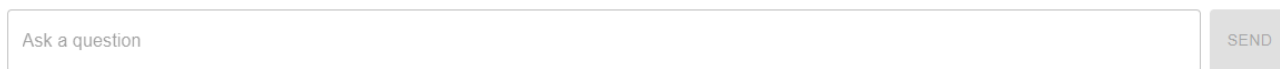


Рисунок 3.3 - Чатовий інтерфейс додатку

Кнопка Send неактивна якщо поле вводу пусте. При натисненні цієї кнопки викликається функція `handleChat` що ховає поле вводу з кнопкою і зберігає попередню відповідь штучного інтелекту та запит користувача в контекст і масив `chat` на основі якого заповнюється динамічний масив текстових контейнерів. Динамічний ReactMarkdown та поле вводу очищаються. Контекст надсилається до штучного інтелекту. Відповідь поступово відображається в динамічному ReactMarkdown після чого знову з'являється поле вводу та кнопка Send.

3.3 Аналіз результатів

Для використання додатку спочатку створюємо аккаунт Hugging Face та отримуємо API ключ згідно наданого посібника. Спочатку оберемо стандарт

PDF/UA-1 та завантажимо файл що відповідає цьому стандарту. Отримуємо підтвердження що він дійсно відповідає обраному стандарту: "PDF file is compliant with Validation Profile requirements". Також оберемо стандарт PDF/UA-2 та завантажимо файл що відповідає цьому стандарту. Отримуємо аналогічний результат.

Потім завантажуюємо третій тестовий файл обравши профіль PDF/UA-2 & ISO 32005. В процесі аналізу користувачу в текстовому вигляді відображається прогрес. Після завершення аналізу користувачу поступово надається відповідь штучного інтелекту створена на основі звіту та системного запиту. Так як штучний інтелект було треновано переважно на англійській мові системний запит також було створено на англійській мові: "Provide a simple explanation of the issues and suggest possible solutions to fix them." Як можемо побачити штучний інтелект перефразував самі проблеми замість пояснення їх важливості(див. Рисунок 3.4).

Certainly! Here's a simple explanation of the issues and possible solutions:

1. TOCI (Table of Contents Item) Identification

Issue: Each TOCI in the table of contents must identify the target of the reference using the Ref entry, either directly on the TOCI structure element itself or on one of its child structure elements. **Solution:** Ensure that every TOCI element has a Ref entry that correctly identifies the target of the reference. If the Ref entry is not directly on the TOCI, it should be on one of its child elements.

2. TOCI and Link

Issue: The TOCI element shall not contain a Link element. **Solution:** Remove any Link elements from the TOCI structure. Links should be handled separately and not nested within TOCI elements.

3. Single Document Structure Element

Issue: The structure tree root shall contain a single Document structure element as its only child. The namespace for that element shall be specified as the PDF 2.0 namespace. **Solution:** Ensure that the structure tree root only has one child, which is a Document structure element. Set the namespace of this element to the PDF 2.0 namespace.

4. Regular Tables

Issue: Tables shall be regular, meaning that table rows shall have the same number of columns (taking into account column spans). **Solution:** Verify that all table rows have the same number of columns, considering any column spans. Adjust the table structure if necessary to ensure uniformity.

5. Figure Properties

Issue: A Figure structure element shall have at least one of the following properties: a) an alternate description (Alt property), or b) a replacement text (ActualText property) that represents the content enclosed by the Figure structure element. **Solution:** Ensure that every Figure element has either an Alt property or an ActualText property. Add the appropriate property if it is missing.

6. PDF/UA Version Specification

Issue: The PDF/UA version of a file shall be specified in the value of the Metadata entry in the document catalog using the PDF/UA identification schema. **Solution:** Include the PDF/UA identification schema in the Metadata entry of the document catalog. Specify the appropriate PDF/UA version.

7. TOCI and Span

Issue: The TOCI element shall not contain a Span element. **Solution:** Remove any Span elements from the TOCI structure. Spans should be handled separately and not nested within TOCI elements.

8. Metadata Stream and dc:title Entry

Issue: The Metadata stream as specified in ISO 32000-2:2020, 14.3 in the document catalog dictionary shall contain a dc:title entry. **Solution:** Ensure that the Metadata stream in the document catalog dictionary includes a dc:title entry. Add the dc:title entry if it is missing.

9. Structure Destinations

Issue: All destinations whose target lies within the current document shall be structure destinations. **Solution:** Ensure that all internal document destinations are structure destinations. This means that each destination should be associated with a structure element in the document.

By addressing these issues, you can ensure that your PDF document meets the required standards and is accessible and well-structured.

Рисунок 3.4 - Результат використання початкового системного запиту

Це не є бажаним результатом тому системний запит був модифікований: "You are an accessibility expert analyzing failed rules from a PDF validator. Provide a list of issues, explain their importance, and suggest solutions." Зазначення ролі та контексту робить відповіді більш сфокусованими та релевантними до задачі. Також завдяки формулюванню запиту штучний інтелект не тільки перефразовує правила але й надає нову інформацію(див. Рисунок 3.5).

Accessibility Issues in PDF Validation

1. Structure Tree Root Issue:

- **Problem:** The structure tree root does not contain a single Document structure element as its only child, and the namespace for that element is not specified as the PDF 2.0 namespace.
- **Importance:** The structure tree is crucial for assistive technologies to navigate and understand the document's logical structure. The PDF 2.0 namespace ensures that the document adheres to the latest standards.
- **Solution:** Ensure that the structure tree root contains a single Document structure element and specify the PDF 2.0 namespace in the Document element.

2. TOCI Reference Issue:

- **Problem:** Each TOCI (Table of Contents Item) in the table of contents does not identify the target of the reference using the Ref entry, either directly on the TOCI structure element itself or on one of its child structure elements.
- **Importance:** Proper references in the TOC are essential for users to navigate directly to the intended sections, especially for those using screen readers.
- **Solution:** Add the Ref entry to each TOCI or its child structure elements to specify the target of the reference.

3. TOCI Content Restriction:

- **Problem:** The TOCI structure elements contain <Link> and elements.
- **Importance:** TOCI elements should be clean and only contain essential information to avoid confusion for assistive technologies.
- **Solution:** Remove <Link> and elements from TOCI structure elements.

4. Table Regularity Issue:

- **Problem:** Tables are not regular; rows do not have the same number of columns, considering column spans.
- **Importance:** Regular tables are easier for assistive technologies to parse and present to users, ensuring that the data is correctly interpreted.
- **Solution:** Ensure that all table rows have the same number of columns, taking into account any column spans.

5. Figure Description Issue:

- **Problem:** Figure structure elements lack an alternate description (Alt property) or a replacement text (ActualText property).
- **Importance:** Descriptions and alternative texts are crucial for users who cannot see the figures, allowing them to understand the content of the figure.
- **Solution:** Add an Alt property or an ActualText property to each Figure structure element to provide a description.

6. Destination Structure Issue:

- **Problem:** Destinations whose target lies within the current document are not structure destinations.
- **Importance:** Structure destinations ensure that links and navigational elements lead to the correct locations within the document, enhancing user experience.
- **Solution:** Convert all internal destinations to structure destinations.

7. PDF/UA Version Specification:

- **Problem:** The PDF/UA version of the file is not specified in the value of the Metadata entry in the document catalog using the PDF/UA identification schema.
- **Importance:** Specifying the PDF/UA version ensures that the document meets the standards for accessibility and can be correctly processed by assistive technologies.
- **Solution:** Add the PDF/UA identification schema to the Metadata entry in the document catalog.

8. Metadata Stream Issue:

- **Problem:** The Metadata stream in the document catalog dictionary does not contain a dc:title entry.
- **Importance:** The dc:title entry provides a title for the document, which is essential for users to identify and understand the content.
- **Solution:** Add a dc:title entry to the Metadata stream in the document catalog dictionary.

Summary

Addressing these issues will significantly improve the accessibility and usability of the PDF document, ensuring that it meets the necessary standards and is easily navigable for all users, including those using assistive technologies.

Рисунок 3.5 - Результат використання модифікованого системного запиту

Перефразування правил може покращити їх зрозумілість але також позбавляє користувача можливості перехресного пошуку правил в інших інструментах, нормативних документах та веб ресурсах. Більш того подібне

представлення результатів неефективно використовує доступний розмір контексту через що системний запит знову було модифіковано: "You are an accessibility expert analyzing failed rules from a PDF validator. Provide a verbatim list of issues, explain their importance, and suggest solutions. Exclude introductions and summaries in your response." Даний запит гарантує збереження формулювання правил що дозволяє зберегти точність використаного аналізатора та зменшити розмір контексту видаливши перший запит користувача. Також цей запит фільтрує непотрібне вступне та заключне речення що також допомагає ефективно використовувати обмежений в розмірі контекст(див. Рисунок 3.6).

**<TOCI> shall not contain **

- **Importance:** The presence of elements within the Table of Contents (TOCI) can lead to reading order issues for assistive technologies, making navigation confusing for users.
- **Solution:** Ensure that the TOCI is structured using only valid elements like <Title> or <H1> to <H6>. Remove or replace elements with appropriate tags that do not disrupt the logical structure of the TOCI.

All destinations whose target lies within the current document shall be structure destinations

- **Importance:** Non-structured destinations can lead to incorrect navigation and accessibility issues, as assistive technologies rely on the logical structure of the document to guide users to specific content.
- **Solution:** Convert all internal destinations to structure destinations by associating them with the correct structure elements in the document's structure tree.

A Figure structure element shall have at least one of the following properties: a) an alternate description (Alt property), as specified in ISO 32000-2:2020, 14.9.3; b) a replacement text (ActualText property) that represents the content enclosed by the Figure structure element

- **Importance:** Figures without alternative text or replacement text cannot be understood by users of assistive technologies, leading to a significant loss of information.
- **Solution:** Add an Alt property to provide a concise description of the figure, or use the ActualText property to include text that replaces the figure content, ensuring that the purpose and content of the figure are accessible.

The structure tree root shall contain a single Document structure element as its only child. The namespace for that element shall be specified as the PDF 2.0 namespace

- **Importance:** A well-structured document is crucial for accessibility, ensuring that the document's content is properly organized and can be navigated logically by assistive technologies.
- **Solution:** Verify that the structure tree root only contains one Document structure element, and that this element is defined in the PDF 2.0 namespace. Adjust the structure tree as necessary to meet this requirement.

<TOCI> shall not contain <Link>

- **Importance:** Links within the TOCI can disrupt the reading order and cause confusion for assistive technology users. The TOCI should be a clean, navigable structure.
- **Solution:** Remove <Link> elements from the TOCI and ensure that all links are properly placed within the main content of the document, where they can be easily navigated and accessed.

Tables shall be regular. Table rows shall have the same number of columns (taking into account column spans)

- **Importance:** Irregular tables can be difficult to navigate for users of assistive technologies, leading to confusion and potential loss of data.
- **Solution:** Ensure that all table rows have the same number of columns, adjusting row spans and column spans as necessary to maintain a consistent structure. Use proper tagging to define table headers and data cells.

The PDF/UA version of a file shall be specified in the value of the Metadata entry in the document catalog using the PDF/UA identification schema

- **Importance:** Specifying the PDF/UA version in the metadata ensures that the document is recognized as accessible, which is crucial for compliance and usability.
- **Solution:** Add the PDF/UA identification schema to the Metadata entry in the document catalog, specifying the correct PDF/UA version (e.g., PDF/UA-1 or PDF/UA-2).

Each TOCI in the table of contents shall identify the target of the reference using the Ref entry, either directly on the TOCI structure element itself or on one of its child structure elements

- **Importance:** Without a Ref entry, the TOCI references will be unusable, as they will not link to the correct sections of the document.
- **Solution:** Ensure that each TOCI entry has a Ref entry that correctly identifies the target section or page in the document. This can be done by adding the Ref entry directly to the TOCI structure element or to one of its child elements.

Рисунок 3.6 - Результат використання остаточного системного запиту

Якість відповіді достатня для поставленої мети тому останній системний запит обрано остаточним. У випадку якщо відповідь недостатньо або занадто детальна для користувача він має можливість продовжити спілкування зі

штучним інтелектом та задати додаткові питання чи вимоги до відповіді(див. Рисунок 3.7).

summarize required actions

Required Actions:

1. **Remove or Replace Elements in TOCI:**
 - Ensure the Table of Contents (TOCI) only uses valid structure elements like <Title> or <H1> to <H6>.
2. **Convert Internal Destinations to Structure Destinations:**
 - Ensure all internal destinations are associated with the correct structure elements in the document's structure tree.
3. **Add Alternative Text or Replacement Text to Figures:**
 - Provide an Alt property or ActualText property to describe or replace the content of figures.
4. **Ensure Document Structure Element in Structure Tree Root:**
 - Verify the structure tree root contains a single Document structure element in the PDF 2.0 namespace.
5. **Remove <Link> Elements from TOCI:**
 - Place all links within the main content of the document.
6. **Make Tables Regular:**
 - Ensure all table rows have the same number of columns, adjusting spans as necessary.
7. **Specify PDF/UA Version in Metadata:**
 - Add the PDF/UA identification schema to the Metadata entry in the document catalog.
8. **Add Ref Entries to TOCI:**
 - Ensure each TOCI entry has a Ref entry identifying the target section or page.
9. **Add dc:title Entry to Metadata:**
 - Include a dc:title entry in the Metadata stream to provide a descriptive title for the document.
10. **Remove Content Items from TOCI:**
 - Ensure the TOCI only contains structural elements and place non-structural content in the main document.

Рисунок 3.7 - Результат додаткової взаємодії користувача зі штучним інтелектом

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено інтелектуальну технологію для оцінки доступності офісних документів. Особлива увага приділена інтеграції технологій штучного інтелекту для підвищення зручності представлення результатів користувачам.

У ході виконання кваліфікаційної роботи магістра було виконано наступні завдання:

1. Виконано аналіз предметної області що показав що розроблювана технологія актуальна та матиме попит серед бізнесу, державних установ і освітніх організацій.
2. Проведено огляд існуючих рішень і стандартів у сфері забезпечення доступності офісних документів на основі якого було сформовано функціональні вимоги до додатку.
3. Спроектовано додаток та створено ряд діаграм нотацій UML та IDEF0.
4. Розроблено веб додаток на основі фреймворку React з використанням бібліотеки Material UI.
5. Реалізовано аналіз доступності документів із застосуванням VeraPDF REST API та VeraPDF Web Server API.
6. Інтегровано засоби штучного інтелекту для гармонізації звіту за допомогою великих мовних моделей сервісу Hugging Face з використанням Serverless Inference API.
7. Обрано тестові дані та проведено аналіз результатів з подальшим вдосконаленням системного запиту.

На основі виконаного дослідження було підтверджено, що розроблений веб-додаток сприяє зменшенню витрат на перевірку доступності документів, підвищує зрозумілість аналізу та надає рекомендації щодо виправлення виявлених проблем. Результати роботи мають практичне значення для організацій, які прагнуть забезпечити відповідність своїх електронних документів сучасним стандартам доступності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. References - Unified Web Site Accessibility Guidelines [Електронний ресурс] // W3C. – Режим доступу: <https://www.w3.org/WAI/GL/central.htm#REF.Ref> (дата звернення: 03.11.2024). – Назва з екрана.
2. Document - Unified Web Site Accessibility Guidelines [Електронний ресурс] // W3C. – Режим доступу: <https://www.w3.org/WAI/GL/central.htm> (дата звернення: 03.11.2024). – Назва з екрана.
3. Web Content Accessibility Guidelines (WCAG) 2.0 [Електронний ресурс] // W3C. – Режим доступу: <https://www.w3.org/TR/2008/REC-WCAG20-20081211/> (дата звернення: 03.11.2024). – Назва з екрана.
4. ISO/IEC 40500:2012 [Електронний ресурс] // ISO. – Режим доступу: <https://www.iso.org/standard/58625.html> (дата звернення: 03.11.2024). – Назва з екрана.
5. Web Content Accessibility Guidelines (WCAG) 2.1 [Електронний ресурс] // W3C. – Режим доступу: <https://www.w3.org/TR/WCAG21/> (дата звернення: 03.11.2024). – Назва з екрана.
6. Web Content Accessibility Guidelines (WCAG) 2.2 [Електронний ресурс] // W3C. – Режим доступу: <https://www.w3.org/TR/WCAG22/> (дата звернення: 13.11.2024). – Назва з екрана.
7. ISO 19005-1:2005 [Електронний ресурс] // ISO. – Режим доступу: <https://www.iso.org/standard/38920.html> (дата звернення: 03.11.2024). – Назва з екрана.
8. ISO 19005-2:2011 [Електронний ресурс] // ISO. – Режим доступу: <https://www.iso.org/standard/50655.html> (дата звернення: 03.11.2024). – Назва з екрана.
9. ISO 19005-3:2012 [Електронний ресурс] // ISO. – Режим доступу: <https://www.iso.org/standard/57229.html> (дата звернення: 03.11.2024). – Назва з екрана.

10. ISO 19005-4:2020 [Электронный ресурс] // ISO. – Режим доступа: <https://www.iso.org/standard/71832.html> (дата звернення: 03.11.2024). – Назва з екрана.
11. ISO 14289-1:2012 [Электронный ресурс] // ISO. – Режим доступа: <https://www.iso.org/standard/54564.html> (дата звернення: 03.11.2024). – Назва з екрана.
12. PDF/UA-1:2014 now published [Электронный ресурс] // PDF Association – PDF’s technical community. – Режим доступа: <https://pdfa.org/pdfua-12014-now-published/> (дата звернення: 03.11.2024). – Назва з екрана.
13. ISO 14289-2 (PDF/UA-2), the “gold standard” for accessibility in PDF 2.0, has arrived [Электронный ресурс] // PDF Association – PDF’s technical community. – Режим доступа: <https://pdfa.org/iso-14289-2-pdf-ua-2-the-gold-standard-for-accessibility-in-pdf-2-0-has-arrived/> (дата звернення: 03.11.2024). – Назва з екрана.
14. ISO 14289-1:2014 [Электронный ресурс] // ISO. – Режим доступа: <https://www.iso.org/standard/64599.html> (дата звернення: 03.11.2024). – Назва з екрана.
15. ISO 14289-2:2024 [Электронный ресурс] // ISO. – Режим доступа: <https://www.iso.org/standard/82278.html> (дата звернення: 03.11.2024). – Назва з екрана.
16. Infographics: PDF/UA and WCAG 2.0 [Электронный ресурс] // PDF Association. – Режим доступа: <https://pdfa.org/infographics-pdfua-and-wcag-2-0/> (дата звернення: 03.11.2024). – Назва з екрана.
17. Acrobat Pro pricing & options [Электронный ресурс] // Adobe. – Режим доступа: <https://www.adobe.com/acrobat/pricing.html> (дата звернення: 03.11.2024). – Назва з екрана.
18. Camden R. Introducing the Adobe PDF Accessibility Checker API Beta [Электронный ресурс] / Raymond Camden // Medium. – Режим доступа: <https://blog.developer.adobe.com/introducing-the-accessibility-checker-beta-api-08ed594ebc8b> (дата звернення: 03.11.2024). – Назва з екрана.

19. PDF Document Solutions [Электронный ресурс] // Allyant. – Режим доступа: <https://allyant.com/pdf-document-solutions/> (дата звернения: 03.11.2024). – Назва з екрана.
20. PDF Services API Pricing [Электронный ресурс] // Adobe Developer. – Режим доступа: <https://udp.adobe.io/document-services/pricing/main/> (дата звернения: 03.11.2024). – Назва з екрана.
21. Validate and Fix PDF Accessibility [Электронный ресурс] // PAVE. – Режим доступа: <https://pave-pdf.org/index.html?lang=en> (дата звернения: 03.11.2024). – Назва з екрана.
22. pdfaPilot [Электронный ресурс] // callas software. – Режим доступа: <https://www.callassoftware.com/en/products/pdfapilot> (дата звернения: 03.11.2024). – Назва з екрана.
23. pdfaPilot Desktop Pricing [Электронный ресурс] // callas software. – Режим доступа: <https://www.callassoftware.com/en/webshop/callas-pdfapilot-desktop> (дата звернения: 03.11.2024). – Назва з екрана.
24. pdfaPilot Server/CLI Pricing [Электронный ресурс] // callas software. – Режим доступа: <https://www.callassoftware.com/en/webshop/callas-pdfapilot-server-cli> (дата звернения: 03.11.2024). – Назва з екрана.
25. European Internet Inclusion Initiative (EIII) [Электронный ресурс] // AGE Platform Europe. – Режим доступа: <https://www.age-platform.eu/project/european-internet-inclusion-initiative/> (дата звернения: 03.11.2024). – Назва з екрана.
26. eAccessibility PDF Tests [Электронный ресурс] // Tingtun Checker. – Режим доступа: <https://checkers.eiii.eu/en/pdftests/> (дата звернения: 03.11.2024). – Назва з екрана.
27. PDF Techniques for WCAG 2.0 [Электронный ресурс] // W3C. – Режим доступа: <https://www.w3.org/TR/2012/NOTE-WCAG20-TECHS-20120103/pdf.html> (дата звернения: 03.11.2024). – Назва з екрана.
28. eAccessibility PDF Check [Электронный ресурс] // Tingtun Checker. – Режим доступа: <https://checkers.eiii.eu/en/pdfcheck/> (дата звернения: 03.11.2024). – Назва з екрана.

29. EIII Project's Tingtun PDF Accessibility Checker Upload Capability Restored [Электронный ресурс] // ACHECKS. – Режим доступа: <https://www.achecks.org/eiii-projects-tingtun-pdf-accessibility-checker-upload-capability-restored/> (дата звернення: 03.11.2024). – Назва з екрана.
30. PAC History [Электронный ресурс] // PDF Accessibility Checker. – Режим доступа: <https://pac.pdf-accessibility.org/en/history> (дата звернення: 03.11.2024). – Назва з екрана.
31. PDF Accessibility Checker [Электронный ресурс] // axesCheck. – Режим доступа: <https://check.axes4.com/en/> (дата звернення: 03.11.2024). – Назва з екрана.
32. Software Products & Services [Электронный ресурс] // GrackleDocs Inc. – Режим доступа: <https://www.grackledocs.com/en/products-services/software/> (дата звернення: 03.11.2024). – Назва з екрана.
33. Desktop GUI Quick Start Guide [Электронный ресурс] // veraPDF Docs. – Режим доступа: <https://docs.verapdf.org/gui/> (дата звернення: 03.11.2024). – Назва з екрана.
34. CLI Quick Start Guide [Электронный ресурс] // veraPDF Docs. – Режим доступа: <https://docs.verapdf.org/cli/> (дата звернення: 03.11.2024). – Назва з екрана.
35. Developing with veraPDF [Электронный ресурс] // veraPDF Docs. – Режим доступа: <https://docs.verapdf.org/develop/> (дата звернення: 03.11.2024). – Назва з екрана.

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

package.json

```
{
  "name": "pdfaxesai",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.13.3",
    "@emotion/styled": "^11.13.0",
    "@huggingface/inference": "^2.8.1",
    "@mui/icons-material": "^6.1.6",
    "@mui/material": "^6.1.6",
    "react": "^18.3.1",
    "react-dom": "^18.3.1",
    "react-markdown": "^9.0.1",
    "spark-md5": "^3.0.2"
  },
  "devDependencies": {
    "@eslint/js": "^9.13.0",
    "@types/react": "^18.3.12",
    "@types/react-dom": "^18.3.1",
    "@vitejs/plugin-react": "^4.3.3",
    "eslint": "^9.13.0",
    "eslint-plugin-react": "^7.37.2",
    "eslint-plugin-react-hooks": "^5.0.0",
    "eslint-plugin-react-refresh": "^0.4.14",
    "globals": "^15.11.0",
    "vite": "^5.4.10"
  }
}
```

vite.config.js

```
import { defineConfig } from 'vite';
import react from '@vitejs/plugin-react';

export default defineConfig({
  plugins: [react()],
  server: {
    proxy: {
      '/api/validate': {
        target: 'https://demo.verapdf.org',
        changeOrigin: true
      },
      '/api/files': {
        target: 'https://pdf4wcag.com/validate',
        changeOrigin: true
      },
      '/api/jobs': {
        target: 'https://pdf4wcag.com/validate',
        changeOrigin: true
      }
    }
  }
});
```

index.html

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>PDFAxesAI</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.jsx"></script>
  </body>
</html>
```

main.jsx

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import App from './App.jsx'
```

```

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <App />
  </StrictMode>,
)

```

App.jsx

```

import {useEffect, useRef, useState} from 'react';
import {
  Box,
  Button,
  Container,
  Dialog,
  DialogActions,
  DialogContent,
  DialogTitle,
  FormControl,
  InputLabel,
  Link,
  MenuItem,
  Select,
  TextField,
  Typography
} from '@mui/material';
import CloudUploadIcon from '@mui/icons-material/CloudUpload';
import ReactMarkdown from 'react-markdown';
import {checkJob, createJob, executeJob, getValidationResult, uploadFile,
validatePdf} from './apiHandler';
import {parseJson} from './jsonHandler.js';
import {getTextGenerationStream} from './aiHandler.js';

function App() {
  const [profile, setProfile] = useState('PDFUA_2_TAGGED_PDF');
  const [model, setModel] = useState('Qwen/Qwen2.5-72B-Instruct');
  const [displayedText, setDisplayedText] = useState('Select a Validation
Profile and upload a file to validate it.');
```

```

const [chat, setChat] = useState([]);
const [inputLock, setInputLock] = useState(true);
const systemPrompt = "You are an accessibility expert analyzing failed rules
from a PDF validator. " +
    "Provide a verbatim list of issues, explain their importance, and suggest
solutions. " +
    "Exclude introductions and summaries in your response.";
const messagesRef = useRef([]);

const handleApiKeyChange = (event) => {
    setApiKey(event.target.value);
};

const handleValidate = () => {
    const regex = /^hf_[a-zA-Z]{34}$/;
    if (regex.test(apiKey)) {
        sessionStorage.setItem('huggingfaceApiKey', apiKey);
        setDialogOpen(false);
    } else {
        setError(true);
        setApiKey('');
    }
};

useEffect(() => {
    const storedApiKey = sessionStorage.getItem('huggingfaceApiKey');
    if (storedApiKey) {
        setDialogOpen(false);
    }
}, []);

const handleProfileChange = (event) => {
    setProfile(event.target.value);
};

const handleModelChange = (event) => {
    setModel(event.target.value);
};

const handleFileChange = async (event) => {
    const file = event.target.files[0];

```

```

    if (file) {
        let json;
        if (['1a', '2a', '3a'].includes(profile)) json = await
validatePdf(file, profile);
        else {
            let jsonResponse = await uploadFile(file);
            const fileId = jsonResponse.id;
            jsonResponse = await createJob(fileId, profile);
            const jobId = jsonResponse.id;
            await executeJob(jobId);
            do {
                jsonResponse = await checkJob(jobId);
                if (jsonResponse.tasks[0].status === 'PROCESSING')
setDisplayText(jsonResponse.progress);
                else if (jsonResponse.tasks[0].status === 'ERROR') throw new
Error(jsonResponse.tasks[0].errorMessage);
                else if (jsonResponse.tasks[0].status === 'CANCELLED') break;
                await new Promise(resolve => setTimeout(resolve, 1000));
            } while (['WAITING',
'PROCESSING'].includes(jsonResponse.tasks[0].status));
            if (jsonResponse.tasks[0].status === 'FINISHED') json = await
getValidationResult(jsonResponse.tasks[0].validationResultId);
            setDisplayedText('');
        }
        const text = parseJson(json);
        if (text === '') setDisplayedText('PDF file is compliant with
Validation Profile requirements.');
```

```

        else {
            const key = sessionStorage.getItem('huggingfaceApiKey');
            messagesRef.current.length = 0;
            messagesRef.current.push({role: 'system', content:
systemPrompt});
            messagesRef.current.push({role: 'user', content: text});
            const stream = getTextGenerationStream(key, model,
messagesRef.current);
            setDisplayedText('');
            for await (const chunk of stream) {
                if (chunk.choices && chunk.choices.length > 0) {
                    appendDisplayedText(chunk.choices[0].delta.content);
                }
            }
        }
    }
}

```

```

        messagesRef.current.pop();
        setInputLock(false);
    }
}
};

const handleChat = async () => {
    setInputLock(true);
    setChat([...chat, displayedText, input]);
    messagesRef.current.push({role: 'assistant', content: displayedText});
    setDisplayedText('');
    messagesRef.current.push({role: 'user', content: input});
    setInput('');
    const key = sessionStorage.getItem('huggingfaceApiKey');
    const stream = getTextGenerationStream(key, model, messagesRef.current);
    for await (const chunk of stream) {
        if (chunk.choices && chunk.choices.length > 0) {
            appendDisplayedText(chunk.choices[0].delta.content);
        }
    }
    setInputLock(false);
};

const appendDisplayedText = (text) => {
    setDisplayedText((prevState) => prevState + text);
};

return (
    <Container>
        <Dialog open={dialogOpen}>
            <DialogTitle>Please enter a valid Hugging Face API
key</DialogTitle>
            <DialogContent>
                <TextField
                    label="Hugging Face API Key"
                    variant="outlined"
                    fullWidth
                    type="password"
                    value={apiKey}
                    onChange={handleApiKeyChange}
                    error={error}
                />
            </DialogContent>
        </Dialog>
    </Container>
);

```

```

        helperText={error ? 'Invalid API key format' : ''}
        margin="normal"
        placeholder="hf_XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
    />
    <Link href="https://huggingface.co/docs/api-
inference/getting-started#getting-a-token"
        target="_blank" rel="noopener">
        Get a token
    </Link>
</DialogContent>
<DialogActions>
    <Button onClick={handleValidate} color="primary"
variant="contained">
        Validate
    </Button>
</DialogActions>
</Dialog>
<FormControl fullWidth>
    <InputLabel id="profile-select-label">Validation
Profile</InputLabel>
    <Select
        labelId="profile-select-label"
        value={profile}
        label="Validation Profile"
        onChange={handleProfileChange}
        variant="outlined"
    >
        <MenuItem value="1a">PDF/A-1A</MenuItem>
        <MenuItem value="2a">PDF/A-2A</MenuItem>
        <MenuItem value="3a">PDF/A-3A</MenuItem>
        <MenuItem value="PDFUA_1">PDF/UA-1</MenuItem>
        <MenuItem value="PDFUA_2">PDF/UA-2</MenuItem>
        <MenuItem value="PDFUA_2_TAGGED_PDF">PDF/UA-2 & ISO
32005</MenuItem>
        <MenuItem value="WTPDF_1_0_ACCESSIBILITY">WTPDF 1.0
Accessibility</MenuItem>
        <MenuItem value="WCAG_2_2_MACHINE">WCAG 2.2
Machine</MenuItem>
        <MenuItem value="WCAG_2_2_COMPLETE">WCAG 2.2 Machine &
Human</MenuItem>
    </Select>

```



```

</FormControl>
<Box mt={2}>
  <FormControl fullWidth>
    <InputLabel id="model-select-label">AI Model</InputLabel>
    <Select
      labelId="model-select-label"
      value={model}
      label="Model"
      onChange={handleModelChange}
      variant="outlined"
    >
      <MenuItem value="Qwen/Qwen2.5-72B-Instruct">Qwen2.5-72B-
Instruct</MenuItem>
      <MenuItem value="meta-llama/Llama-3.1-70B-
Instruct">Llama-3.1-70B-Instruct</MenuItem>
      <MenuItem value="mistralai/Mixtral-8x7B-Instruct-
v0.1">Mixtral-8x7B-Instruct-v0.1</MenuItem>
      <MenuItem value="google/gemma-2-9b-it">gemma-2-9b-
it</MenuItem>
    </Select>
  </FormControl>
</Box>
<Box mt={2}>
  <Button component="label" variant="contained"
startIcon={<CloudUploadIcon />} fullWidth>
    Upload file
    <input type="file" accept="application/pdf" hidden
onChange={handleFileChange} />
  </Button>
</Box>
<Box mt={2}>
  {chat.map((item, index) => (
    index % 2 === 0 ?
      (<ReactMarkdown key={index}>{item}</ReactMarkdown>) :
      (<Typography key={index}
align="right">{item}</Typography>))
  )}
</Box>
<ReactMarkdown>{displayedText}</ReactMarkdown>
<Box mt={2} display="flex" justifyContent="space-between">
  <TextField

```

```

        fullWidth
        variant="outlined"
        value={input}
        onChange={(e) => setInput(e.target.value)}
        style={{ display: inputLock ? 'none' : 'block' }}
        placeholder="Ask a question"
      />
      <Button
        variant="contained"
        color="primary"
        onClick={handleChat}
        style={{ display: inputLock ? 'none' : 'block', marginLeft:
'8px' }}
        disabled={!input.trim()}
      >
        Send
      </Button>
    </Box>
  </Container>
);
}

```

```
export default App;
```

apiHandler.js

```

import SparkMD5 from 'spark-md5';
export const validatePdf = async (file, profileId) => {
  const endpoint = `/api/validate/${profileId}`;
  const formData = new FormData();
  formData.append('file', file);

  const headers = {
    'X-File-Size': file.size,
    'Accept': 'application/json'
  };

  const response = await fetch(endpoint, {
    method: 'POST',
    headers: headers,
    body: formData
  });
};

```

```

    if (!response.ok) {
      throw new Error('Failed to validate PDF');
    }

    return response.json();
  };

export const uploadFile = async (file) => {
  const endpoint = '/api/files';
  const formData = new FormData();
  formData.append('file', file);
  const contentMD5 = await calculateContentMD5(file);
  formData.append('contentMD5', contentMD5);

  const headers = {
    'Accept': 'application/json'
  };

  const response = await fetch(endpoint, {
    method: 'POST',
    headers: headers,
    body: formData
  });

  if (!response.ok) {
    throw new Error('Failed to upload file');
  }

  return response.json();
};

const calculateContentMD5 = file =>
  new Promise(resolve => {
    const CHUNK_SIZE = 2 * 1024 * 1024;
    const CHUNKS = Math.ceil(file.size / CHUNK_SIZE);
    const blobSlice = File.prototype.slice || File.prototype.mozSlice ||
File.prototype.webkitSlice;
    const spark = new SparkMD5.ArrayBuffer();
    const reader = new FileReader();
    let currentChunk = 0;

```

```

const loadNext = () => {
  const start = currentChunk * CHUNK_SIZE;
  const end = start + CHUNK_SIZE >= file.size ? file.size : start +
CHUNK_SIZE;

  reader.readAsArrayBuffer(blobSlice.call(file, start, end));
};

reader.onload = event => {
  spark.append(event.target.result);
  currentChunk++;
  currentChunk < CHUNKS ? loadNext() : resolve(spark.end());
};

loadNext();
});

export const createJob = async (fileId, profile) => {
  const endpoint = '/api/jobs';

  const headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json'
  };

  const body = JSON.stringify({
    profile: profile,
    tasks: [
      {
        fileId: fileId
      }
    ]
  });

  const response = await fetch(endpoint, {
    method: 'POST',
    headers: headers,
    body: body
  });

  if (!response.ok) {

```

```
        throw new Error('Failed to create job');
    }

    return response.json();
};

export const executeJob = async (jobId) => {
    const endpoint = `/api/jobs/${jobId}/execution`;

    const headers = {
        'Content-Type': 'application/json',
        'Accept': 'application/json'
    };

    const response = await fetch(endpoint, {
        method: 'POST',
        headers: headers
    });

    if (!response.ok) {
        throw new Error('Failed to execute job');
    }

    return response.json();
};

export const checkJob = async (jobId) => {
    const endpoint = `/api/jobs/${jobId}`;

    const headers = {
        'Accept': 'application/json'
    };

    const response = await fetch(endpoint, {
        method: 'GET',
        headers: headers
    });

    if (!response.ok) {
        throw new Error('Failed to check job status');
    }
}
```

```

    return response.json();
  };

export const getValidationResult = async (fileId) => {
  const endpoint = `/api/files/${fileId}`;

  const response = await fetch(endpoint, {
    method: 'GET'
  });

  if (!response.ok) {
    throw new Error('Failed to get validation result');
  }

  return response.json();
};

```

jsonHandler.js

```

export const parseJson = (json) => {
  let validationResult;
  if (json.report && json.report.jobs && json.report.jobs[0] &&
  json.report.jobs[0].validationResult) {
    validationResult = json.report.jobs[0].validationResult;
  } else {
    validationResult = json;
  }
  if (validationResult.details && validationResult.details.ruleSummaries &&
  validationResult.details.ruleSummaries.length > 0) {
    const descriptions = validationResult.details.ruleSummaries.map(summary =>
  summary.description);
    return descriptions.join('\n');
  }
  return '';
};

```

aiHandler.js

```

import {HfInference} from "@huggingface/inference";

export const getTextGenerationStream = (apiKey, model, messages) => {
  const inference = new HfInference(apiKey);
  return inference.chatCompletionStream({

```

```
    model: model,  
    messages: messages,  
    max_tokens: 2048  
  });  
}
```