МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інтелектуальних інформаційних систем

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____»_____ 2026 р.

КВАЛІФІКАЦІЙНА РОБОТА

НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ВЕБЗАСТОСУНОК ДЛЯ АВТОМАТИЗАЦІЇ ДОКУМЕНТООБІГУ НА ОСНОВІ ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ

Спеціальність 122 Комп'ютерні науки

Освітня програма «Комп'ютерні науки»

Здобувач _____ Анастасія ГОЛУБКОВА

«___»_____ 2026 р.

Керівник канд. техн. наук _____ Віктор ГОЖИЙ

«___»_____ 2026 р.

Миколаїв – 2026

Чорноморський національний університет імені Петра Могили
(повне найменування закладу вищої освіти)

Факультет	Комп’ютерних наук
Кафедра	Інтелектуальних інформаційних систем
Рівень вищої освіти	Перший (бакалаврський)
Освітній ступень	Бакалавр
Спеціальність	122 Комп’ютерні науки
Освітня програма	Комп’ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри інтелектуальних інформаційних систем

_____ Євген СІДЕНКО

«____» _____ 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу здобувача

Голубкової Анастасії Сергіївни
((прізвище, ім’я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Вебзастосунок для автоматизації документообігу на основі використання інтелектуальних технологій».

Керівник роботи: Гожий Віктор Олександрович, старший викладач кафедри інтелектуальних інформаційних систем, канд. техн. наук.

Затверджена наказом ЧНУ ім. Петра Могили від «25» грудня 2025 р. № 353.

2. Строк представлення кваліфікаційної роботи «___» _____ 2025 р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні: вебзастосунок Crewdeck ERP для автоматизованого вилучення та структурованого збереження відомостей із документів членів екіпажів суден на основі технологій OCR; технічне завдання на систему, архітектура клієнт-серверної взаємодії, UML-діаграми та макети інтерфейсу у Figma; програмна реалізація серверної частини на

Python/Flask, клієнтської частини на React.js, модулів OCR та шаблонної екстракції даних.

4. Перелік питань, що підлягають розробці: аналіз предметної галузі та визначення функціональних вимог до системи; огляд наукових публікацій та аналіз існуючих програмних аналогів; порівняльний аналіз технологій та програмних засобів реалізації; розроблення технічного завдання, проєктування UML-діаграм та макетів інтерфейсу; програмна реалізація серверної та клієнтської частини; інтеграція OCR-модуля на основі Tesseract та реалізація модуля шаблонної екстракції даних; проведення тестування та усунення виявлених дефектів.

5. Перелік графічних матеріалів: презентація.

Керівник роботи _____ Віктор ГОЖИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Анастасія ГОЛУБКОВА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата видачі завдання «22» грудня 2026 р.

КАЛЕНДАРНИЙ ПЛАН
кваліфікаційної роботи

Тема: Вебзастосунок для автоматизації документообігу на основі використання інтелектуальних технологій

№	Найменування роботи	Початок	Закінчення	Примітки
1	Отримання завдання на виконання КР	21.12.2025	24.12.2025	Виконано
2	Аналіз предметної області та постановка задачі	25.12.2025	30.01.2026	Виконано
3	Огляд літературних джерел за темою кваліфікаційної роботи, зокрема огляд публікацій та комерційних аналогів	31.01.2026	01.03.2026	Виконано
4	Розроблення технічного завдання, проєктування архітектури системи, UML-діаграм та макетів інтерфейсу	02.03.2026	01.04.2026	Виконано
5	Програмна реалізація серверної та клієнтської частин застосунку, інтеграція OCR-модуля та модуля шаблонної екстракції, тестування системи та аналіз отриманих результатів	02.04.2026	24.05.2026	Виконано
6	Перший попередній захист КР на засіданні комісії кафедри	25.05.2026	25.05.2026	Виконано
7	Корегування роботи за результатами попереднього захисту	26.05.2026	04.06.2026	Виконано
8	Другий попередній захист КР на засіданні комісії кафедри	05.06.2026	05.06.2026	Виконано
9	Доробка та остаточне оформлення КР	06.06.2026	14.06.2026	Виконано
10	Подання КР, її електронної копії та інших документів (відгуку, рецензії) до захисту	15.06.2026	19.06.2026	Виконано

Керівник роботи _____ Віктор ГОЖИЙ
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Здобувач _____ Анастасія ГОЛУБКОВА
(Особистий підпис) *(Власне ім'я ПРІЗВИЩЕ)*

Дата складання календарного плану
«29» січня 2026 р.

АНОТАЦІЯ

до кваліфікаційної роботи
здобувачки групи 401з ЧНУ ім. Петра Могили

Голубкової Анастасії Сергіївни

на тему: **“ВЕБЗАСТОСУНОК ДЛЯ АВТОМАТИЗАЦІЇ ДОКУМЕНТООБІГУ НА ОСНОВІ ВИКОРИСТАННЯ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ”**

Актуальність роботи полягає у необхідності автоматизації обробки кадрової документації членів екіпажів суден у морській галузі з використанням технологій оптичного розпізнавання символів і шаблонної екстракції, що дозволить скоротити час обробки документів та знизити ймовірність помилок.

Об’єктом дослідження є процес обробки кадрової документації членів екіпажів суден у морській галузі.

Предметом дослідження є програмні засоби та алгоритми структурованого вилучення інформації з документів із застосуванням технологій OCR та шаблонного аналізу.

Метою роботи є розробка вебзастосунку Crewdeck ERP для автоматизованого вилучення й збереження відомостей із документів моряків.

В результаті виконання роботи розроблено вебзастосунок що включає OCR-модуль на основі Tesseract, модуль шаблонної екстракції на основі регулярних виразів, модуль управління профілями моряків та модуль експорту у XLSX. Практична цінність полягає у скороченні часу обробки документів з кількох годин до хвилин. Точність OCR перевищує 95%, точність шаблонної екстракції – 89%.

Дана робота складається з трьох розділів: аналіз предметної галузі та вибір технологій; проектування архітектури, бази даних та інтерфейсу; програмна реалізація та тестування.

Сторінок – 94, таблиць – 13, рисунків – 13, додатків – 3, джерел – 31.

Ключові слова: оптичне розпізнавання символів, шаблонна екстракція, документообіг, морська галузь, Flask, React.js, MongoDB, вебзастосунок, REST API, Tesseract.

ABSTRACT

to the qualification work by the student of the group 401z of Petro Mohyla Black Sea National University

Holubkova Anastasiia

“ WEB APPLICATION FOR DOCUMENT WORKFLOW AUTOMATION BASED ON INTELLIGENT TECHNOLOGIES”

The relevance of this work lies in the need to automate personnel documentation processing in the maritime industry using optical character recognition and template extraction technologies, which will reduce processing time and minimize data entry errors.

The object of research is the process of handling personnel documentation of vessel crew members in maritime industry organizations.

The subject of research encompasses software tools and algorithms for structured data extraction from documents using OCR and template-based analysis technologies.

The aim is to develop the Crewdeck ERP web application for automated extraction and organized storage of crew member document data.

As a result, the Crewdeck ERP web application was developed, including a Tesseract-based OCR module, a template extraction module using regular expressions, a seamen profile management module, and an XLSX export module. The practical value lies in reducing document processing time from several hours to minutes. OCR accuracy exceeds 95%, template extraction accuracy reaches 89%.

This work consists of three sections: subject domain analysis and technology selection; architecture, database and interface design; software implementation and testing.

Pages – 94, tables – 13, figures – 13, appendices – 3, references – 31.

Key words: optical character recognition, template extraction, document workflow, maritime industry, Flask, React.js, MongoDB, web application, REST API, Tesseract.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – інтерфейс програмування застосунків

BCrypt – алгоритм хешування паролів

CRUD – базові операції з даними

CSS – каскадні таблиці стилів

DPI – кількість точок на дюйм

ERP – планування ресурсів підприємства

HTTP – протокол передачі гіпертексту

HTTPS – захищений протокол передачі гіпертексту

JSON – текстовий формат обміну даними

JWT – токен для автентифікації

MLC – конвенція про працю в морському судноплавстві

MVP – мінімально життєздатний продукт

NoSQL – клас нереляційних баз даних

OCR – оптичне розпізнавання символів

PDF – формат переносного документа

REST – архітектурний стиль взаємодії компонентів

SQL – мова структурованих запитів

STCW – Міжнародна конвенція про підготовку і дипломування моряків

UML – уніфікована мова моделювання

URL – уніфікований локатор ресурсів

БД – база даних

ІС – інформаційна система

ПЗ – програмне забезпечення

ТЗ – технічне завдання

ВСТУП

Автоматизація документообігу є актуальною задачею для багатьох галузей, проте у морській сфері вона набуває особливого значення. Кожен член екіпажу зобов’язаний мати актуальний пакет документів відповідно до вимог міжнародних конвенцій STCW та MLC 2006, а судноплавні компанії та кадрові агентства щомісяця обробляють сотні таких пакетів переважно вручну. Відсутність доступного спеціалізованого рішення що поєднувало б автоматичне розпізнавання даних з документів та зручний інструмент управління базою моряків і зумовила вибір теми кваліфікаційної роботи – розробка вебзастосунку Crewdeck ERP для автоматизації документообігу із застосуванням інтелектуальних технологій розпізнавання тексту (OCR на основі нейронних мереж) та шаблонної екстракції даних.

У ході виконання роботи проведено аналіз предметної галузі та наявних систем-аналогів, обґрунтовано вибір технологій, розроблено технічне завдання та спроєктовано архітектуру системи, реалізовано серверну та клієнтську частини застосунку з інтеграцією OCR-модуля та модуля шаблонної екстракції, проведено тестування та задокументовано розроблений застосунок.

Робота складається з трьох розділів. Перший містить аналіз предметної галузі та порівняльний аналіз технологій. Другий описує проєктування системи – технічне завдання, архітектуру, діаграми та макети інтерфейсу. Третій розділ присвячено програмній реалізації, тестуванню та керівництву користувача.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ВИБІР ТЕХНОЛОГІЙ

1.1 Постановка задачі

Кожен моряк, що виходить у рейс, повинен мати актуальний комплект документів відповідно до вимог конвенцій STCW [9] та MLC 2006 [8]: диплом, медичну довідку, паспорт моряка, сертифікати з безпеки та ряд інших. Судноплавні компанії та кадрові агентства обробляють сотні таких комплектів щомісяця, і переважна більшість цієї роботи – ручне введення даних з документів у корпоративні системи. Це не лише трудомістко, а й створює ризик помилок, які в умовах міжнародного морського права можуть мати реальні правові наслідки.

Наявні на ринку універсальні системи обробки документів або не адаптовані під специфіку морської галузі, або є надто дорогими для середніх за розміром агенств. Спеціалізовані морські ERP-системи, своєю чергою, не мають вбудованого інструменту автоматичного вилучення даних. Все це підтверджує актуальність розробки спеціалізованого доступного рішення.

Метою кваліфікаційної роботи є розробка вебзастосунку Crewdeck ERP для автоматизації процесу вилучення та структурованого збереження даних з документів у морській галузі на основі технологій OCR та шаблонної екстракції.

Об'єктом дослідження є процес автоматизованої обробки кадрової документації членів екіпажів у морській галузі.

Предметом дослідження є методи, алгоритми та програмні засоби інтелектуальної екстракції структурованих даних з документів різних форматів із застосуванням технологій OCR та шаблонного розпізнавання.

Для досягнення зазначеної мети вирішено такі завдання: проведено аналіз предметної галузі та визначено функціональні вимоги; розроблено технічне завдання; спроєктовано архітектуру системи та структуру бази даних; розроблено UI/UX дизайн у Figma для трьох ключових екранів застосунку; реалізовано серверну частину на Python/Flask; реалізовано клієнтську частину на React.js; інтегровано OCR-модуль на основі Tesseract; реалізовано модуль шаблонної

екстракції; проведено тестування на трьох рівнях та усунуто виявлені дефекти; задокументовано розроблений застосунок.

1.2 Термінологічна база дослідження

Перш ніж переходити до опису системи та технологій, доцільно визначити ключові поняття, якими оперує дана робота.

Документообіг – це сукупність процесів, пов'язаних зі створенням, отриманням, передачею, зберіганням та використанням документів в організації. У морській галузі під документообігом розуміють насамперед роботу з персональними документами членів екіпажу: дипломами, сертифікатами, контрактами, медичними довідками та іншими паперами, наявність і актуальність яких є обов'язковою вимогою міжнародних конвенцій.

Автоматизація документообігу – передбачає впровадження програмних засобів, що дозволяють виконувати операції з документами: введення даних, класифікацію, маршрутизацію, зберігання без участі людини або з мінімальним її втручанням. Ступінь автоматизації може бути різним: від розпізнавання окремих полів до повністю безлюдного наскрізного процесу обробки.

OCR (Optical Character Recognition) – технологія оптичного розпізнавання символів, яка перетворює зображення тексту (скановані документи, фотографії) у машинозчитуваний текстовий формат. Сучасні OCR-системи, особливо ті, що використовують нейронні мережі, здатні розпізнавати текст з точністю понад 99% на документах належної якості.

У контексті даної роботи саме нейромережевий компонент OCR-системи (LSTM-рушій) розглядається як інтелектуальна технологія, що забезпечує адаптивне розпізнавання тексту незалежно від шрифту та якості сканування, на відміну від класичних методів розпізнавання за жорсткими шаблонами символів.

Шаблон вилучення даних – це набір правил, що визначають, які поля потрібно знайти в документі та де саме вони розташовані на сторінці. Шаблони дають можливість обробляти великі серії однотипних документів автоматично:

система один раз налаштовується на конкретний тип документа і надалі сама знаходить потрібні дані без ручного втручання.

ERP-система (Enterprise Resource Planning) – інтегрована система управління підприємством, що об’єднує в одному середовищі ключові бізнес-процеси: управління персоналом, фінансами, логістикою, документообігом. У морській галузі, як правило, ERP-системи містять модулі управління екіпажами, сертифікаційного контролю та кадрового планування.

Вебзастосунок – програмний застосунок, що виконується на веб-сервері і доступний користувачу через браузер. На відміну від десктопних програм, вебзастосунок не потребує встановлення і однаково доступний з будь-якого пристрою з підключенням до мережі.

MongoDB – документно-орієнтована СУБД класу NoSQL, що зберігає дані у форматі JSON-подібних документів з гнучкою схемою. Ця особливість робить її зручним вибором для зберігання різнорідних даних, структура яких може відрізнятися від запису до запису.

REST API (Representational State Transfer Application Programming Interface) – архітектурний підхід до організації взаємодії між клієнтською та серверною частинами застосунку через HTTP-протокол. Забезпечує стандартизований та незалежний від платформи інтерфейс обміну даними.

Drag-and-drop – механізм взаємодії користувача з інтерфейсом, що дозволяє переміщати об’єкти (зокрема файли) шляхом перетягування в певну зону екрана. Суттєво спрощує процес завантаження файлів у систему.

Figma – хмарний інструмент для проєктування інтерфейсів і створення прототипів, що підтримує спільну роботу кількох учасників у реальному часі. Широко застосовується для розробки макетів до початку програмної реалізації продукту.

1.3 Огляд наукових публікацій та комерційних аналогів

Тематика інтелектуальної обробки документів активно досліджується як в академічному, так і в індустрії. Серед існуючих інтелектуальних систем проаналізовано шість рішень, що найближче стоять до задачі проєкту.

Adobe Acrobat Pro DC є одним із найпоширеніших інструментів для роботи з PDF-документами [21]. Продукт підтримує вилучення тексту, базове розпізнавання форм та конвертацію файлів. Водночас система не призначена для пакетної обробки великих серій документів і не передбачає налаштовуваних шаблонів вилучення. Щомісячна підписка й орієнтація на офісних користувачів-аналістів, а не на галузеві потреби, роблять цей інструмент лише допоміжним засобом, але не системним рішенням для морських агенств.

Docparser [18] – хмарна платформа для автоматичного вилучення даних з документів. Підтримує налаштовувані правила парсингу для PDF і дає змогу налаштувати правила вилучення без програмування. Попри зручний інтерфейс, платформа не має галузевої специфіки і не надає можливості управління профілями персоналу. Вартість підписки зростає залежно від кількості оброблених документів.

Rossum.ai [19] – платформа когнітивного захоплення даних на основі нейронних мереж, орієнтована передусім на обробку рахунків-фактур та фінансових документів. Демонструє високу точність при роботі з типовими форматами, проте потребує хмарного підключення і не адаптована для роботи зі специфічними морськими документами (сертифікати STCW, паспорти моряка тощо).

Hyperscience [20] – корпоративна платформа інтелектуальної автоматизації, що поєднує машинне навчання з інструментами перевірки даних оператором. Орієнтована на великі організації з тисячами документів на місяць і відповідним рівнем витрат. Для середнього кадрового агентства такий рівень складності та вартості є невиправданим.

CrewInspector [22] – спеціалізована система для морської галузі, що дозволяє відстежувати сертифікати екіпажу та контролювати терміни їх дії. Проте вона не має вбудованого модуля автоматичного розпізнавання та вилучення даних, усі дані вносяться вручну. Продукт закриває лише задачу контролю, але не автоматизацію введення.

Compas (Crewing Management Software) [23] – представник класу спеціалізованих морських ERP-систем. Охоплює широкий спектр функцій кадрового управління: розклади екіпажу, кадровий облік, сертифікаційний контроль. Разом з тим, подібно до CrewInspector, не має вбудованого інструменту OCR або шаблонного вилучення даних з документів.

Таблиця 1.1 – Порівняльний аналіз існуючих систем за ключовими критеріями

Система	OCR / вилучення даних	Шаблонна екстракція	Галузева специфіка (морська)	Управління профілями	Цінова доступність
Adobe Acrobat Pro	Часткова	Ні	Ні	Ні	Середня
Docparser	Так	Так	Ні	Ні	Середня
Rossum.ai	Так (ШІ)	Обмежена	Ні	Ні	Висока
Hyperscience	Так (ШІ)	Так	Ні	Ні	Дуже висока
CrewInspector	Ні	Ні	Так	Так	Середня
Compas	Ні	Ні	Так	Так	Висока
Crewdeck ERP (розробляється)	Так	Так	Так	Так	Низька

Наведений аналіз підтверджує, що жодне з розглянутих рішень не поєднує інтелектуального вилучення даних зі спеціалізацією під морську галузь та прийнятною вартістю впровадження. Саме цю нішу покликаний заповнити застосунок Crewdeck ERP.

Нижче наведено огляд публікацій та комерційних систем, що мають пряме відношення до теми роботи.

У дослідженні Sharma та співавторів (2022), присвяченому застосуванню трансформерних моделей для вилучення інформації з документів [1], показано, що попередньо навчені моделі на кшталт LayoutLM значно перевершують класичні підходи при роботі зі складноструктурованими документами – формами, рахунками, таблицями. Автори підкреслюють, що для практичних систем найефективнішим залишається комбінований підхід: спочатку OCR для отримання тексту, потім модель для його структурування. Цей висновок підтверджує доцільність обраного в роботі комбінованого підходу: нейромережевий OCR для розпізнавання тексту з подальшим структуруванням результату.

Dengel та Klein (2020) систематизували методи автоматичного розпізнавання та класифікації ділових документів [2] – від простих алгоритмів на основі ключових слів до нейромережевих класифікаторів. Їх висновок про оптимальність гібридного підходу для корпоративних систем (класифікація типу документа з подальшим застосуванням правил вилучення) узгоджується з підходом, реалізованим у модулі шаблонів системи Crewdeck ERP, де структурування виконується після етапу інтелектуального розпізнавання тексту.

Робота Chiticariu (2018) описує систему IBM SystemT [3] – одну з перших промислових платформ для вилучення інформації з текстів на основі декларативних правил. Принципи модульності та налаштовуваних правил, що були закладені в цій системі, стали індустріальним стандартом і знайшли відображення в сучасних комерційних рішеннях.

Окремої уваги заслуговують дослідження, присвячені безпосередньо обробці документів у галузях з високим рівнем регуляторних вимог, до яких належить і морська.

Robaldo (2019) [4] розглядали питання автоматичного аналізу юридичних і регуляторних текстів та дійшли висновку, що стандартні методи вилучення інформації потребують суттєвої адаптації при роботі з документами, що мають

жорстку юридичну структуру. Для морської галузі це особливо актуально: сертифікати STCW та документи відповідно до MLC 2006 мають специфічні поля та форми, які відрізняються залежно від країни видачі, але при цьому несуть однотипну за змістом інформацію. Це підтверджує необхідність гнучкої шаблонної системи, а не універсального підходу до розпізнавання.

У роботі Katti (2018) [5] запропоновано метод Chargrid – двовимірне представлення документа на рівні символів для подальшої обробки згортковими нейронними мережами. Підхід показав значно кращі результати порівняно з суто текстовими методами при роботі з рахунками-фактурами та формами, оскільки враховує просторове розташування тексту на сторінці. Цей принцип, брати до уваги не лише зміст, а й розташування текстових блоків, є методологічним обґрунтуванням для модуля шаблонної екстракції в системі Crewdeck ERP, де пошук значень полів здійснюється за ключовими словами, що передують шуканому значенню у тексті документа.

Варто також згадати роботу Xu (2020) [6], в якій представлено модель LayoutLMv2, вдосконалену версію LayoutLM з підтримкою візуальних ознак документа поряд з текстовими та позиційними. Автори продемонстрували результати, що перевищують попередній стан речей на кількох бенчмарках розуміння документів. Хоча застосування повноцінних трансформерних моделей виходить за межі поточного проєкту через ресурсоємність, розуміння цього напряму є важливим для визначення перспектив розвитку системи.

1.4 Порівняльний аналіз технологій та програмних засобів

Технології оптичного розпізнавання символів. Tesseract OCR [7, 10] – відкрита бібліотека, що розробляється за підтримки Google і підтримує понад 100 мов, включаючи кирилицю. У версії 4.x до базового алгоритму додано нейромережевий рушій на базі LSTM, що суттєво підвищило точність розпізнавання рукописного та низькоякісного сканованого тексту. Для Python доступна обгортка pytesseract, яка робить інтеграцію бібліотеки в застосунок достатньо простою. Tesseract обрано як

основний OCR-систем для даного проєкту – насамперед через відкритий код, відсутність витрат на ліцензування та достатню точність для роботи з типовими документами моряків.

Google Cloud Vision API та Amazon Textract є хмарними альтернативами з вищою точністю розпізнавання і вбудованою підтримкою структурованого вилучення даних з форм і таблиць. Разом з тим, вони передбачають постійне підключення до хмари і поіменну оплату за оброблені документи, що робить їх менш прийнятними для початкової реалізації системи.

Бібліотеки для роботи з документами. PyMuPDF (fitz) [11] – Python-бібліотека на основі MuPDF для роботи з PDF. Забезпечує швидке вилучення тексту, зображень, метаданих та координат текстових блоків на сторінці. Є одним з найпродуктивніших рішень у Python-екосистемі для цієї задачі.

python-docx та openpyxl використовуватимуться для читання документів у форматах DOCX та XLSX відповідно, а також для генерації вихідних файлів при експорті результатів.

Серверні технології. Python 3.11 обрано основною мовою програмування бекенду. Обґрунтування очевидне: саме Python має найбагатшу екосистему бібліотек для обробки документів, зображень і даних, що є критично важливим для даного проєкту.

Flask [12] – мікрофреймворк для розробки вебзастосунків на Python. На відміну від Django, він не нав’язує жорсткої структури і дає змогу підключати лише ті компоненти, які справді потрібні. Це зручно для побудови REST API з поступовим нарощуванням функціоналу.

Обробка документів виконується синхронно у межах одного запиту до серверної частини. При зростанні навантаження архітектура системи передбачає можливість переходу до асинхронної обробки через чергу завдань, проте для поточного етапу розробки синхронний підхід є достатнім.

Клієнтські технології. React.js [13] обрано для реалізації фронтенду. Компонентна архітектура бібліотеки добре підходить для побудови складних

інтерфейсів з динамічним оновленням даних, зокрема, для відображення прогресу обробки документів у реальному часі.

База даних та інфраструктура. MongoDB [14] обрано через гнучку схему даних. Структура документів моряків може суттєво відразнятися залежно від типу та країни видачі, тому документно-орієнтована СУБД є природнішим вибором, ніж реляційна.

Docker [17] використовуватиметься для контейнеризації кожного компонента системи, що спрощує розгортання і гарантує однакове середовище виконання незалежно від сервера.

Завантажені файли зберігаються локально у директорії uploads/ серверної частини застосунку. При необхідності масштабування архітектура системи дозволяє замінити локальне сховище на хмарне рішення без суттєвої переробки коду.

Figma використовується для проєктування інтерфейсу до початку програмної реалізації. Дозволяє швидко перевіряти та узгоджувати дизайнерські рішення без написання коду.

Порівняльний аналіз OCR-рушіїв. Оскільки OCR є одним з ключових компонентів системи, вибір рушія потребував окремого порівняльного аналізу. Розглянуто три основні варіанти: Tesseract OCR, Google Cloud Vision API та Amazon Textract.

Tesseract є повністю відкритим рішенням без жодних витрат на ліцензування. Підтримує понад 100 мов і добре справляється з якісними сканами. Головний його недолік – точність помітно падає на документах низької якості або зі складним макетом. Крім того, «з коробки» він не вміє розпізнавати структуру таблиць – для цього потрібні додаткові бібліотеки.

Google Cloud Vision API та Amazon Textract є хмарними платними сервісами корпоративного рівня. Обидва демонструють вищу точність розпізнавання, особливо на складних документах, і мають вбудовану підтримку структурованого вилучення даних з форм і таблиць. Amazon Textract, зокрема, повертає координати

кожного знайденого текстового блоку, що могло б спростити реалізацію шаблонної екстракції. Проте обидва сервіси передбачають оплату за кожну оброблену сторінку та вимагають постійного підключення до хмари, що робить їх менш прийнятними для початкової реалізації проєкту.

З огляду на наведені характеристики, для поточного етапу розробки обрано Tesseract як основний OCR-рушій. Це рішення обґрунтовується відсутністю витрат на ліцензіонування, можливістю локального розгортання та достатньою точністю для роботи з типовими документами моряків належної якості сканування. У подальшому архітектура системи дозволяє замінити або доповнити Tesseract хмарним сервісом без суттєвої переробки коду, завдяки тому, що модуль OCR виділено в окремий компонент.

Таблиця 1.2 – Порівняльна характеристика OCR-рушіїв

Критерій	Tesseract OCR	Google Cloud Vision API	Amazon Textract
Модель розповсюдження	Відкритий код (Apache 2.0)	Хмарний сервіс (платний)	Хмарний сервіс (платний)
Вартість	Безкоштовно	Від $1,5 за 1000 зображень	Від $1,5 за 1000 сторінок
Підтримка мов	100+ мов, у т.ч. кирилиця	50+ мов	Англійська та обмежений набір
Точність на якісних сканах	Висока (>98%)	Дуже висока (>99%)	Дуже висока (>99%)
Точність на низькоякісних документах	Середня	Висока	Висока
Розпізнавання таблиць	Потребує додат. Бібліотек	Часткова підтримка	Вбудована підтримка
Повернення координат блоків	Так (hOCR/TSV)	Так	Так
Локальне розгортання	Так	Ні (хмара)	Ні (хмара)

Кінець таблиці 1.2

Залежність від мережі	Немає	Обов’язкова	Обов’язкова
Інтеграція з Python	pytesseract	google-cloud-vision	boto3
Рекомендований варіант для MVP	Обрано	–	–

Порівняльний аналіз Flask та Django. При виборі серверного фреймворку розглядалися два найпоширеніші варіанти для Python, такі як, Flask та Django.

Django є повнофункціональним фреймворком з вбудованою ORM, адміністративною панеллю, системою автентифікації та багатьма іншими готовими компонентами. Він добре підходить для великих монолітних застосунків, де важлива швидкість старту розробки. Однак для даного проєкту такий підхід є надлишковим: вбудована ORM Django орієнтована на реляційні бази даних і не дає жодних переваг при роботі з MongoDB. Крім того, жорстка структура Django ускладнює побудову легкого REST API без зайвих залежностей.

Flask є мікрофреймворком, він надає лише мінімально необхідний набір інструментів і не нав’язує жодних рішень щодо структури проєкту чи вибору компонентів. Це дає змогу підключати лише те, що справді потрібно, і будувати архітектуру так, як цього вимагає конкретна задача. Для побудови REST API з поступовим нарощуванням функціоналу Flask є природнішим вибором. Додатково, Flask добре інтегрується з PyMongo та бібліотеками для обробки документів, що є важливим для даного проєкту.

Таблиця 1.3 – Порівняння фреймворків Flask та Django за критеріями проєкту

Критерій	Flask	Django
Тип фреймворку	Мікрофреймворк	Повнофункціональний фреймворк
Вбудована ORM	Ні	Так (лише для реляційних БД)

Кінець таблиці 1.3

Сумісність з MongoDB	Через PyMongo (нативно)	Через сторонні бібліотеки
Жорсткість структури проєкту	Гнучка	Жорстка (MVS-шаблон)
Вбудована адмін-панель	Ні	Так
Надлишковість для REST API	Низька	Висока
Підходить для даного проєкту	Обрано	–

Робота з MongoDB через PyMongo. PyMongo [15] є офіційним Python-драйвером для роботи з MongoDB. Він надає низькорівневий, але зручний інтерфейс для виконання всіх операцій з базою даних: вставки, пошуку, оновлення та видалення документів. На відміну від ORM-підходу, PyMongo працює безпосередньо з Python-словниками, що є природним способом роботи з JSON-подібними документами MongoDB.

Для підключення до бази даних використовується клас MongoClient, після чого доступ до конкретної колекції здійснюється через прості звернення виду db.collection_name. Операції пошуку підтримують гнучкі запити з фільтрами, сортуванням та агрегацією. Важливою особливістю є підтримка індексів, наприклад, індекс на поле owner_id у колекціях seamen та templates суттєво прискорить вибірку даних конкретного користувача при великій кількості записів.

Організація середовища через Docker Compose. Для оркестарції всіх компонентів застосунку вирішено використати Docker Compose. Практична цінність цього інструменту для проєкту полягає в тому, що він дає змогу описати всю інфраструктуру, Flask-сервер та MongoDB, у єдиному файлі docker-compose.yml Завдяки цьому вся складна система стартує буквально однією командою.

Працює це таким чином: кожна частина застосунку піднімається у власному ізольованому контейнері. Вони «спілкуються» між собою через закриту внутрішню

мережу, тому назовні не відкриваються жодні зайві порти, що є великим плюсом для безпеки. Головна перевага такого підходу це максимальна простота розгортання. Тобто, взагалі не важливо, яка операційна система стоїть на цільовому сервері чи які там встановлені бібліотеки. Єдина вимога для успішного запуску це наявність самого Docker.

Сценарії використання системи. Для кращого розуміння того, як система працюватиме на практиці, рамках технічного завдання було описано основні сценарії використання. Три найбільш показових:

Сценарій 1: Оператор завантажує пакет медичних довідок моряків

Оператор кадрового відділу отримує від агента пакет із двадцяти сканованих медичних довідок у форматі PDF. Замість того щоб відкривати кожен файл вручну і переносити дані в таблицю, він відкриває сторінку «Управління даними моряків», перетягує всі файли одразу у зону drag-and-drop і натискає кнопку обробки. Система автоматично застосовує до кожного файлу заздалегідь створений шаблон «Медичний сертифікат», розпізнає прізвище, ім’я, дату видачі та термін дії довідки і додає отримані записи до бази. Якщо якийсь файл не вдалося розпізнати коректно, система позначає його в журналі зі статусом «Помилка» і оператор може обробити його окремо вручну.

Сценарій 2: Адміністратор створює новий шаблон для паспорта моряка

До агентства починають надходити паспорти моряків нового зразка з дещо зміненим розташуванням полів. Існуючий шаблон вже не підходить. Адміністратор переходить на сторінку «Шаблони екстракції», натискає «+ Створити шаблон» і задає назву нового шаблону та його категорію. Далі він вказує перелік полів, які потрібно вилучати: прізвище, ім’я, по батькові, номер документа, дата народження, дата видачі, орган видачі. Для кожного поля задаються координати його розташування на сторінці документа. Після збереження шаблон стає доступним усім операторам організації і система починає автоматично застосовувати його до нових завантажень.

Сценарій 3: Менеджер експортує дані для передачі судновласнику

Судновласник запитує список членів екіпажу з діючими сертифікатами у форматі Excel. Менеджер переходить на сторінку «Управління даними моряків» та натискає кнопку «Експорт XLSX», після чого система формує файл з усіма профілями та пропонує його завантажити. Весь процес займає менше хвилини замість кількох годин ручного складання таблиці.

Вимоги до безпеки та ролі користувачів. Окремим пунктом технічного завдання опрацьовано виоги до безпеки системи та розмежування прав доступу.

Система передбачає дві ролі користувачів. Роль «Адміністратор» надає повний доступ до всіх функцій: управління профілями моряків, створення та редагування шаблонів, перегляд журналу обробки всіх користувачів організації, управління обліковими записами. Роль «Користувач» дає доступ до завантаження документів, перегляду та використання наявних шаблонів і перегляду власної історії обробки, але без можливості змінювати шаблони або переглядати дані інших користувачів.

Щодо захисту даних, паролі користувачів зберігатимуться виключно у вигляді хешів з використанням алгоритму bcrypt, тобто навіть адміністратор бази даних не матиме доступу до реальних паролів. Автентифікація реалізовуватиметься через JWT-токени (JSON Web Tokens): після успішного входу користувач отримує токен, який додається до кожного подальшого запиту до API і підтверджує його особу та права доступу. Термін дії токена обмежений, що мінімізує ризики у разі його перехоплення.

Усі файли, що завантажуються в систему, проходять перевірку типу та розміру ще до початку обробки. Система відхиляє файли, що не відповідають дозволеним форматам, або ті, розмір яких перевищує встановлений ліміт. Це захищає від потенційних атак через завантаження шкідливих файлів.

Висновки до розділу 1

У першому розділі проведено комплексний аналіз предметної галузі та досліджено проблематику документообігу в організаціях морської сфери. Встановлено, що кадрові агентства та судноплавні компанії опрацьовують значні обсяги персональної документації членів екіпажів переважно у ручному режимі, що суттєво збільшує трудові витрати та підвищує ймовірність виникнення помилок при введенні даних.

Порівняльний аналіз шести наявних програмних рішень підтвердив відсутність на ринку доступного інструменту, який поєднував би вбудований модуль автоматичного розпізнавання документів із галузевою спрямованістю на морську сферу. Проведений огляд технологій дозволив обґрунтувати вибір Tesseract OCR – рушія розпізнавання на основі нейромережевого LSTM-алгоритму, що виступає інтелектуальною складовою системи, а також Flask як серверного фреймворку, React.js для клієнтської частини та MongoDB як бази даних із гнучкою схемою зберігання, що разом утворюють оптимальний стек для реалізації поставленої задачі.

2 ПРАКТИЧНІ РЕЗУЛЬТАТИ ВИКОНАННЯ ЗАВДАНЬ

2.1 Технічне завдання на систему Crewdeck ERP

Першим практичним результатом стало створення детального технічного завдання (ТЗ) на вебзастосунок Crewdeck ERP. Робота над ТЗ дозволила чітко окреслити межі проєкту та зрозуміти, як саме система вирішуватиме проблему ручної обробки документів.

Загальний опис системи. Crewdeck ERP проєктується як надійний інструмент для судноплавних компаній та крюїнгових агенств. Головна ідея полягає в тому, щоб автоматизувати вилучення даних з типових документів моряків (PDF, DOCX, XLSX тощо) та надійно їх зберігати. Система працюватиме на базі двох основних модулів. Перший, «Управління даними моряків», стане єдиним вікном для управління профілями моряків, де до кожного профілю прив’язані відповідні файли. Другий модуль, «Шаблони екстракції», дасть змогу користувачам самостійно створювати шаблони для різних форм документів, вказуючи системі, де саме на сторінці шукати потрібну інформацію.

Функціональні вимоги. Щоб система дійсно була корисною, виділено ряд критичних функціональних вимог (пріоритет – високий):

– робота з файлами (FR-01): користувачі зможуть завантажити одразу кілька файлів (PDF, DOCX, XLSX), використовуючи зручний механізм drag-and-drop;

– вилучення даних (FR-02): застосунок автоматично розпізнає та вилучає текстовий вміст з документів, у тому числі зі сканованих файлів за допомогою OCR-модуля;

– шаблонізація (FR-05, FR-06): це ядро системи. Користувач зможе створити шаблон (наприклад, для медичного сертифіката), вказати ключові слова для пошуку значень полів (ім’я, дата, ранг) і система автоматично застосовуватиме ці правила до всіх нових схожих документів;

– експорт та OCR (FR-08, FR-11): оскільки багато морських документів

є просто сканами, впровадження OCR (через Tesseract) є обов'язковим. Знайдені дані можна вивантажити у формат XLSX для подальшої роботи з даними;

– безпека (FR-09): доступ до платформи здійснюватиметься виключно через механізми автентифікації для захисту конфіденційних даних;

– також передбачено ведення історії обробки (FR-10), щоб кожен користувач міг відстежити статуси своїх завантажень.

Для зручності сприйняття функціональні вимоги систематизовано у таблиці 2.1.

Таблиця 2.1 – Функціональні вимоги до системи Crewdeck ERP

Код вимоги	Назва вимоги	Опис	Пріоритет
FR-01	Завантаження файлів	Підтримка drag-and-drop для PDF, DOCX, XLSX; одночасне завантаження кількох файлів	Високий
FR-02	Вилучення тексту	Автоматичне розпізнавання та вилучення текстового вмісту з документів	Високий
FR-03	Управління профілями	Створення, редагування та видалення профілів моряків із прив'язкою документів	Високий
FR-04	Створення шаблонів	Можливість самостійно створювати шаблони вилучення з визначенням ключових слів для пошуку значень	Високий
FR-05	Застосування шаблонів	Автоматичне застосування шаблону до завантажених документів відповідної категорії	Високий

Кінець таблиці 2.1

Код вимоги	Назва вимоги	Опис	Пріоритет
FR-06	Статуси обробки	Відображення поточного стану обробки кожного документа у реальному часі	Середній
FR-07	Експорт даних	Вивантаження результатів у форматі XLSX	Середній
FR-08	Автентифікація	Доступ до системи лише через авторизація; JWT-токени; розмежування ролей	Високий
FR-09	Журнал обробки	Зберігання та перегляд історії оброблених документів для кожного користувача	Середній
FR-10	OCR-розпізнавання	Інтеграція Tesseract для обробки сканованих документів без машинозчитуваного тексту	Високий

Нефункціональні вимоги. Окрім функціоналу, важливо було закласти вимоги до якості роботи системи. Наприклад, продуктивність має бути такою, щоб обробка стандартного 10-сторінкового документа займала не більше 30 секунд (NFR-01). Система проєктується з розрахунком на безперебійну роботу (доступність 99.5%, NFR-02) та можливість горизонтального масштабування (NFR-03). Окрему увагу приділено безпеці: весь трафік шифруватиметься по HTTPS, а файли зберігатимуться у захищеному середовищі (NFR-04). Звісно, все це має працювати в сучасних браузерах (Chrome, Firefox, Safari, Edge) та мати інтуїтивно зрозумілий інтерфейс (NFR-05, NFR-06).

Таблиця 2.2 – Нефункціональні вимоги до системи Crewdeck ERP

Код	Категорія	Вимога
NFR-01	Продуктивність	Обробка стандартного 10-сторінкового документа не більше 30 секунд
NFR-02	Доступність	Безперебійна робота системи з показником доступності не менше 99,5%
NFR-03	Масштабованість	Підтримка горизонтального масштабування через контейнеризацію (Docker)
NFR-04	Безпека	Шифрування трафіку (HTTPS), хешування паролів (bcrypt), захищене файлове сховище
NFR-05	Сумісність	Коректна робота в актуальних версіях Chrome, Firefox, Safari, Edge
NFR-06	Зручність використання	Виконання основних операцій без спеціального навчання; інтуїтивний інтерфейс

Етапи розробки. Увесь процес створення системи Crewdeck ERP структуровано у вигляді п’яти послідовних фаз, кожна з яких має чітко визначені цілі та результати. Такий підхід забезпечує контрольованість розробки та дозволяє своєчасно виявляти відхилення від плану.

Фаза 1. Аналіз, проєктування та прототипування

Перша фаза є поточною і охоплює три паралельні напрями роботи. По-перше, проводиться детальний аналіз предметної галузі: вивчаються типи документів, з якими працюють морські кадрові агентства, їх структура, обов’язкові поля та формати. По-друге, на основі зібраних вимог проєктується архітектура системи: визначається розподіл на мікросервіси, структура бази даних MongoDB, схема взаємодії між компонентами через REST API. По-третє, у середовищі Figma розробляються макети ключових екранів застосунку, що дозволяє узгодити інтерфейсні рішення до початку написання коду. Результатом фази є затверджене технічне завдання, схема архітектури та набір UI-макетів, готових до реалізації.

Фаза 2. Розробка базової версії – MVP.

На другій фазі реалізується ядро системи: завантаження документів, базове вилучення тексту з PDF-файлів за допомогою PyMuPDF, управління профілями моряків та автентифікація користувачів на основі JWT-токенів. Також розгортається інфраструктура проєкту: налаштовується середовище виконання, підключається база даних MongoDB. Метою фази є отримання працюючого прототипу, на якому можна перевірити основні сценарії використання.

Фаза 3. Розробка модуля шаблонів та інтеграція OCR

Третя фаза є найбільш технічно складною. На цьому етапі реалізується ключовий функціонал системи – модуль шаблонної екстракції, що дозволяє користувачам самостійно визначати поля для вилучення через графічний інтерфейс із зазначенням ключових слів для пошуку у тексті документа. Паралельно інтегрується Tesseract OCR для обробки сканованих документів. Завершується реалізація модуля експорту даних у форматі XLSX та модуля прив’язки документів до профілів моряків. За результатами фази система набуває повного функціоналу, описаного у технічному завданні.

Фаза 4. Тестування та усунення дефектів.

Четверта фаза повністю присвячена забезпеченню якості продукту. Проводиться модульне тестування OCR-модуля та модуля шаблонної екстракції, інтеграційне тестування всіх API-ендпоінтів за допомогою інструменту Postman, а також функціональне тестування основних сценаріїв використання системи. Виявлені дефекти фіксуються та усуваються. Фаза завершується після успішного проходження всіх критичних тест-кейсів.

Фаза 5. Реліз та технічна підтримка

На завершальній фазі система розгортається у робочому середовищі. Формується документація для кінцевих користувачів, зокрема керівництво користувача. Після релізу здійснюється відстеження працездатності системи та збір зворотного зв’язку для планування подальшого розвитку застосунку.

2.2 Архітектура системи та структура бази даних

Загальна архітектура. Щоб система працювала швидко і не «зависала» під час обробки важких PDF-файлів, обрано клієнт-серверну архітектуру з синхронною обробкою документів. Взаємодія між клієнтом (фронтендом) та сервером (бекендом) відбувається через RESTful API по захищеному протоколу HTTPS.

Клієнтська частина застосунку реалізовується з використанням бібліотеки React.js, що забезпечує компонентну структуру інтерфейсу та ефективне оновлення відображуваних даних.

Обробка документів виконується безпосередньо у межах API-запиту після збереження файлу на диск. Після отримання запиту сервер послідовно викликає OCR-модуль для вилучення тексту та модуль шаблонної екстракції для структурування даних. Такий синхронний підхід є достатнім для поточного етапу розробки та забезпечує простоту розгортання системи. При необхідності обробки великих обсягів документів архітектура дозволяє перейти до асинхронної черги завдань без суттєвої переробки бізнес-логіки.

Структура бази даних. MongoDB обрано не випадково, адже вона ідеально підходить для зберігання гнучних структур, таких як шаблони та результати екстракції. База складатиметься з таких основних колекцій:

– users: зберігатиме дані користувачів (_id, email, хеш пароля, роль, дата створення);

– PDFFiles / seamen: для збереження метаданих завантажених файлів та прив’язки їх до профілів моряків (filename, шлях у сховищі, статус обробки, ID власника);

– templates: колекція шаблонів, де в JSON-форматі зберігатимуться правила, координати зон для вилучення даних та категорія документа;

– extractionResults (processing_history): зберігатиме вже структуровані результати парсингу, прив’язані до конкретного файлу.

Технологічний стек:

– бекенд: python 3.11 + Flask;

– обробка документів: PyMuPDF (швидке вилучення тексту), pytesseract (для OCR), python-docx, openpyxl;

– фронтенд: React.js;

– база даних та сховище: MongoDB;

– інфраструктура: Docker для контейнеризації.

Повний технологічний стек системи Crewdeck ERP систематизовано у таблиці 2.3.

Таблиця 2.3 – Технологічний стек системи Crewdeck ERP

Рівень системи	Технологія / Бібліотека	Призначення
Бекенд	Python 3.11 + Flask	Основна мова програмування; серверний фреймворк для REST API
Вилучення тексту з PDF	PyMuPDF (fitz)	Швидке вилучення тексту та координат текстових блоків
OCR-розпізнавання	pytesseract (Tesseract 5.x)	Обробка сканованих документів і перетворення зображень у текст
Робота з DOCX/XLSX	python-docx, openpyxl	Читання та генерація документів Microsoft Office
Фронтенд	React.js + Tailwind CSS	Реалізація клієнтського інтерфейсу
База даних	MongoDB	Зберігання профілів, шаблонів і результатів екстракції

Кінець таблиці 2.3

Файлове сховище	Локальна директорія uploads/	Зберігання завантажених файлів на сервері
Контейнеризація	Docker / Docker Compose	Ізоляція компонентів і спрощення розгортання
UI-прототипування	Figma	Проєктування макетів інтерфейсу до початку розробки

2.2.1 Детальний опис колекцій бази даних

База даних crewdeck складається з чотирьох основних колекцій, кожна з яких відповідає за зберігання певного типу даних системи. Нижче наведено детальний опис структури кожної колекції та призначення її полів.

Колекція users призначена для зберігання облікових записів користувачів системи. Кожен документ колекції містить такі поля: унікальний ідентифікатор _id, що генерується MongoDB автоматично; поле email, яке є унікальним у межах колекції та використовується як логін; поле password, що містить хеш пароля, згенерований алгоритмом bcrypt; поле role, що визначає рівень доступу користувача і може приймати одне з двох значень – «admin» або «user»; поле created_at, що фіксує дату та час створення облікового запису. Структуру документа колекції users наведено у таблиці 2.4.

Таблиця 2.4 – Структура документа колекції users

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор (автоматично)
email	String	Електронна пошта користувача (унікальна)
password	String	Хеш пароля (bcrypt)
role	String	Роль: «admin» або «user»
created_at	Date	Дата та час реєстрації

Колекція seamen є центральною колекцією системи та призначена для зберігання профілів моряків. Кожен профіль містить персональні дані члена екіпажу, а також посилання на завантажені документи. Структуру документа колекції seamen наведено у таблиці 2.5.

Таблиця 2.5 – Структура документа колекції seamen

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
full_name	String	Повне ім’я моряка
rank	String	Ранг або посада на судні
birth_date	String	Дата народження
document_number	String	Номер основного документа
status	String	Статус: «доступний», «на борту», «недоступний»
owner_id	ObjectId	Посилання на користувача-власника запису
files	Array	Масив ідентифікаторів прив’язаних файлів
created_at	Date	Дата створення профілю

Поле status використовує кольорове кодування в інтерфейсі: зелений колір відповідає статусу «доступний», жовтий – «на бору», синій – «недоступний», що відповідає макету сторінки «Управління даними моряків», розробленому у Figma.

Колекція files зберігає метадані завантажених документів. Важливо зазначити, що самі файли зберігаються локально у директорії uploads/, тоді як у базі даних зберігається лише інформація про них. Структуру документа колекції files наведено у таблиці 2.6.

Таблиця 2.6 – Структура документа колекції files

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
filename	String	Оригінальна назва файлу

Кінець таблиці 2.6

filepath	String	Шлях до файлу у директорії uploads/
file_type	String	MIME-тип файлу
file_size	Integer	Розмір файлу у байтах
status	String	Статус обробки: «обробляється», «завершено», «помилка»
owner_id	ObjectId	Посилання на користувача що завантажив файл
seaman_id	ObjectId	Посилання на профіль моряка
uploaded_at	Date	Дата та час завантаження

Поле status відображає поточний стан обробки документа та оновлюється системою автоматично в процесі виконання операцій OCR та шаблонної екстракції.

Колекція templates зберігає шаблони вилучення даних, що створюються користувачами системи. Ключовою особливістю є поле fields, яке є масивом об’єктів – кожен об’єкт описує одне поле для вилучення та містить його назву і перелік ключових слів для пошуку у тексті документа. Структуру документа колекції templates наведено у таблиці 2.7.

Таблиця 2.7 – Структура документа колекції templates

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
name	String	Назва шаблону
category	String	Категорія документа (загальний, медичний, сертифікат, паспорт)
fields	Array	Масив об’єктів з описом полів для вилучення
owner_id	ObjectId	Посилання на користувача-власника
created_at	Date	Дата створення шаблону

Колекція extraction_results зберігає результати шаблонної екстракції для кожного обробленого файлу. Структуру документа колекції extraction_results наведено у таблиці 2.8.

Таблиця 2.8 – Структура документа колекції extraction_results

Поле	Тип	Опис
_id	ObjectId	Унікальний ідентифікатор
file_id	ObjectId	Посилання на оброблений файл
template_id	ObjectId	Посилання на використаний шаблон
extracted_data	Object	Об’єкт з вилученими полями та їх значеннями
confidence	Float	Відсоток успішно заповнених полів
created_at	Date	Дата та час створення результату

Поле confidence обчислюється як відношення кількості успішно знайдених полів до загальної кількості полів шаблону та виражається у відсотках. Цей показник відображається оператору в інтерфейсі та дозволяє швидко оцінити якість розпізнавання без перегляду кожного поля окремо.

2.2.2 Індексування колекцій

Для забезпечення прийнятної швидкості вибірки даних при зростанні обсягу бази передбачено індексування ключових полів колекцій. У колекції users створюється унікальний індекс на поле email – це одночасно прискорює пошук при авторизації та гарантує відсутність дублікатів. У колекціях seamen, files та extraction_results створюються індекси на поле owner_id, оскільки переважна більшість запитів до цих колекцій фільтрує дані за конкретним користувачем. У колекції files додатково індексується поле status для прискорення вибірки документів за станом обробки.

2.3 Проєктування інтерфейсу застосунку у Figma

У процесі проєктування системи Crewdeck ERP розроблено макети трьох ключових екранів застосунку у середовищі Figma. Оскільки цільова аудиторія системи це співробітники кадрових відділів та менеджери судноплавних компаній, інтерфейс спроєктовано з акцентом на функціональність та зручність щоденної роботи з великими обсягами даних. Загальна концепція дизайну базується на мінімалістичному підході зі світлою кольоровою схемою та стриманою навігаційною панеллю ліворуч.

Під час проєктування застосовано принцип ієрархії інформації: найбільш важливі елементи керування розміщено у верхній частині кожного екрана, основний контент займає центральну зону, а допоміжні дії винесено на рівень окремих елементів таблиці або картки. Такий підхід мінімізує кількість кроків для виконання типових операцій та скорочує час навчання нових користувачів роботі з системою. Типографіка інтерфейсу базується на шрифті Poppins, що забезпечує високу читабельність на екранах різних розмірів та відповідає концепції професійного бізнес-застосунку.

Навігаційна панель застосунку відображається на всіх сторінках та містить три основні розділи: «Моряки», «Шаблони» та «Історія». Активний розділ виділяється візуально для орієнтації користувача у структурі застосунку. У нижній частині панелі розміщено ім’я та роль поточного авторизованого користувача, а також кнопку виходу із системи. Така організація навігації забезпечує постійний доступ до всіх розділів системи з будь-якого екрана без необхідності повернення на головну сторінку.

Макет сторінки «Управління даними моряків» є центральним екраном системи та призначений для управління базою профілів членів екіпажу. У верхній частині сторінки розміщено рядок пошуку за іменем або рангом моряка, фільтри за рангом та статусом, а також кнопку «+ Додати моряка» для створення нового профілю. Безпосередньо під панеллю пошуку розташована зона перетягування

файлів з іконкою завантаження та написом «Перетягніть документи моряків сюди». Така зона є ключовим елементом інтерфейсу що реалізує механізм drag-and-drop завантаження документів без додаткових діалогових вікон.

Основну частину екрана займає таблиця з даними моряків. Кожен рядок таблиці містить такі елементи: аватар або ініціали моряка, повне ім’я, статус із кольоровим бейджем, ранг або посада на судні, назва та кількість прив’язаних файлів, а також кнопки видалення запису. Відображення прив’язаного файлу безпосередньо у рядку таблиці дозволяє оператору одразу бачити які документи вже завантажені до профілю кожного моряка. Статуси моряків використовують кольорове кодування для швидкої візуальної оцінки без необхідності читати текст кожного запису. Макет сторінки «Управління даними моряків» представлено на рисунку 2.1.

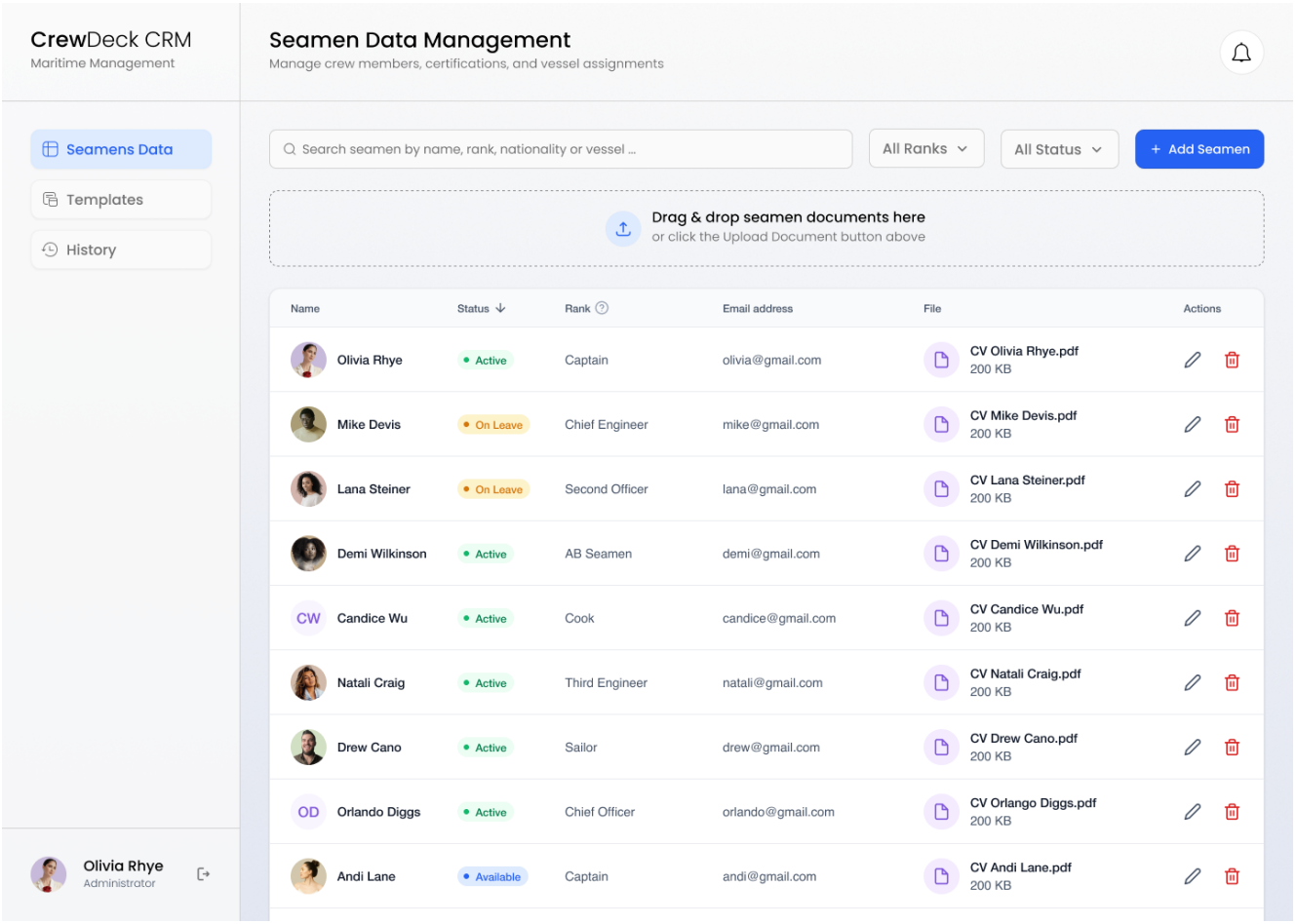

Рисунок 2.1 – Макет сторінки «Управління даними моряків» у Figma

Макет сторінки «Шаблони екстракції» призначений для управління шаблонами вилучення даних з документів. У верхній частині сторінки розміщено рядок пошуку шаблонів, фільтр за категорією, сортування за датою останньої зміни та кнопку «+ Створити шаблон». Шаблони відображаються картками у сітку по три в рядку. Карткове відображення обрано замість табличного оскільки шаблони є описовими об’єктами де важливо одразу бачити назву, категорію та призначення.

Кожна картка містит іконку документа, назву шаблону, категорійну мітку, короткий перелік полів що вилучаються та кнопку видалення. Категорії відповідають типам документів морської галузі: загальний, медичний, сертифікат, паспорт. Наявність кнопки копіювання дозволяє швидко створювати нові шаблони на основі наявних без необхідності заповнювати форму з нуля. Макет сторінки «Шаблони екстракції» представлено на рисунку 2.2.

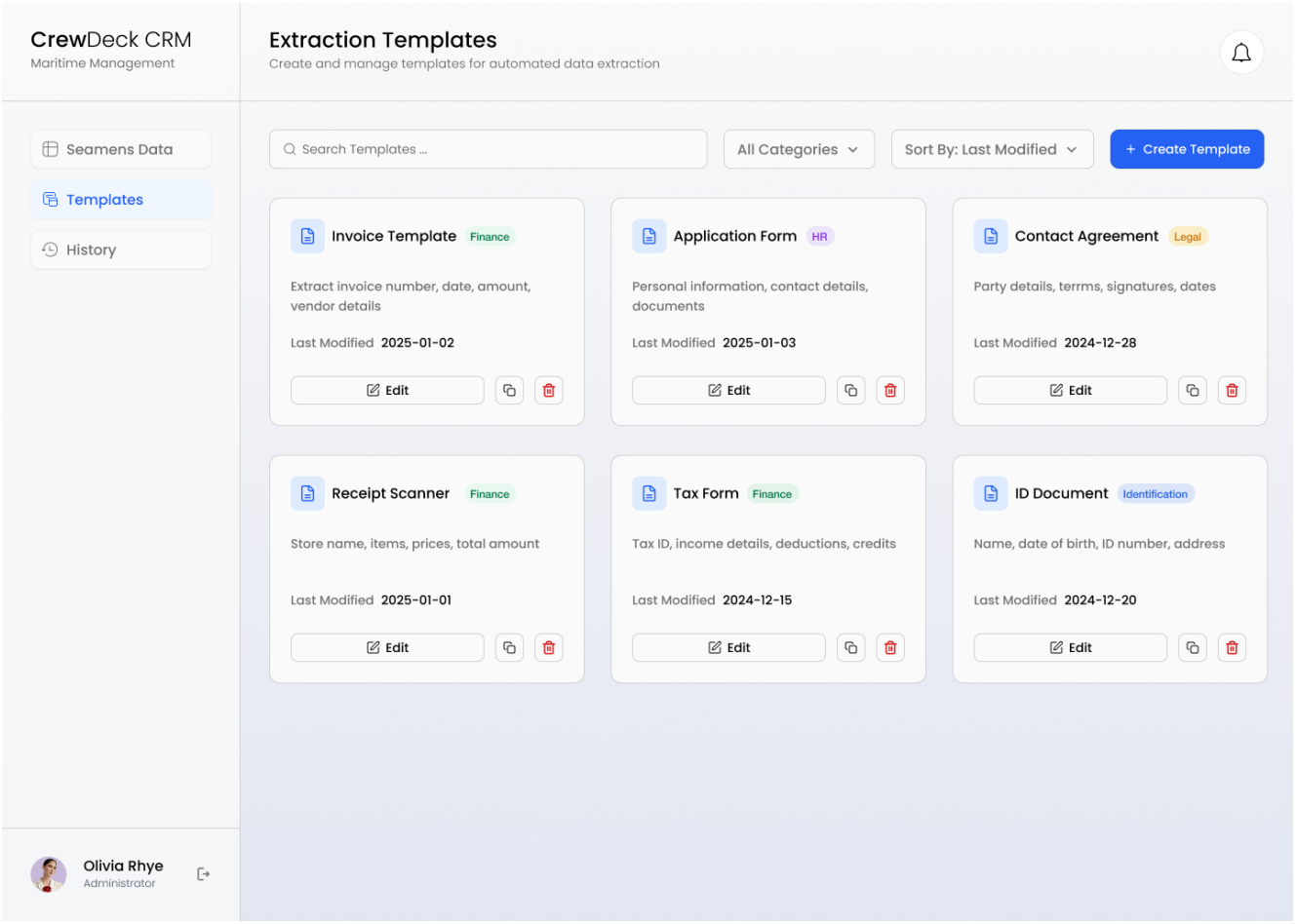

Рисунок 2.2 – Макет сторінки «Шаблони екстракції» у Figma

Макет сторінки «Історія обробки» призначений для перегляду журналу всіх операцій завантаження та обробки документів у системі. У верхній частині сторінки розміщено рядок пошуку, фільтр за датою є особливо важливою для операторів що обробляють великі обсяги документів щодня, вона дозволяє швидко знайти записи за конкретний робочий день.

Основну частину екрана займає часова шкала подій де кожен запис оформлено у вигляді окремої картки. Картка події містить іконку типу документа, заголовок події, ім’я користувача що виконав дію, назву обробленого файлу, короткий опис події, точний час виконання операції та статус обробки. Формат часової шкали замість звичайної таблиці обрано оскільки хронологічна послідовність подій є більш природною моделлю для журналу операцій, а картковий формат дозволяє відображати більше інформації про кожну подію. Макет сторінки «Історія обробки» представлено на рисунку 2.3.

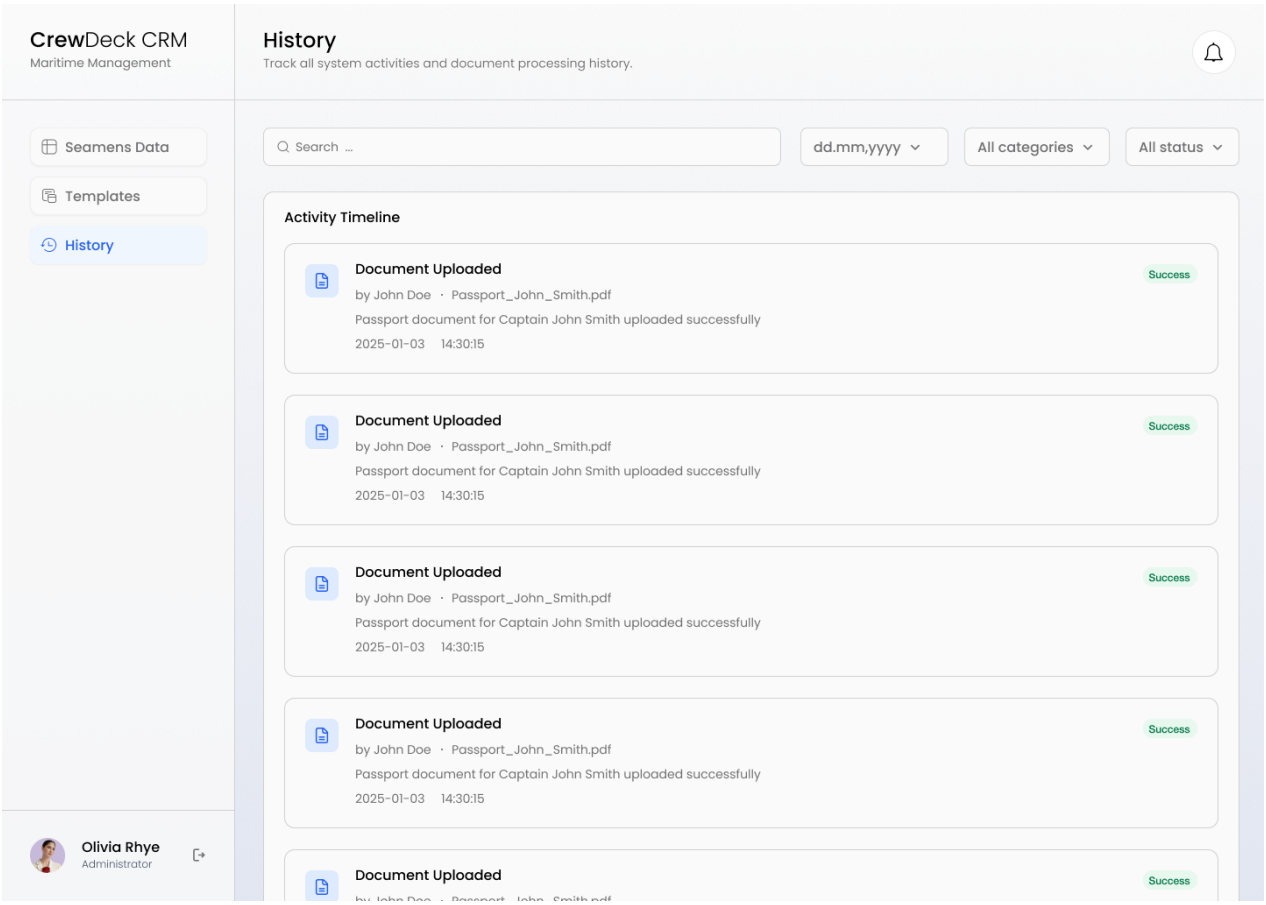

Рисунок 2.3 – Макет сторінки «Історія обробки» у Figma

Розроблені макети трьох екранів системи, «Управління даними моряків», «Шаблони екстракції» та «Історія обробки», формують цілісну дизайн-систему застосунку та визначають загальний візуальний напрям програмної реалізації. Єдина типографіка та принципи розміщення елементів забезпечують узгодженість інтерфейсу на всіх сторінках. Спроєктований інтерфейс охоплює всі основні сценарії використання системи: управління профілями моряків з прив'язкою документів, налаштування шаблонів вилучення даних та моніторинг журналу обробки.

2.4 Моделювання основного бізнес-процесу системи

Для нормалізації функціонування системи Crewdeck ERP розроблено модель основного бізнес-процесу обробки документів. Моделювання дозволило чітко розмежувати операції, що виконуються системою автоматично, від тих, що залишаються за оператором, а також виявити потенційні точки відмови та передбачити механізми їх обробки.

Основний процес розпочинається з моменту надходження пакету документів. Оператор завантажує файли через інтерфейс drag-and-drop, після чого система перевіряє формат і розмір кожного файлу. У разі невідповідності вимогам файл відхиляється із відповідним повідомленням. Файли, що пройшли валідацію, зберігаються локально, а їх метадані записуються до колекції files бази даних MongoDB зі статусом processing.

На наступному етапі система визначає спосіб вилучення тексту. Якщо PDF-файл містить машинозчитуваний текст, він вилучається безпосередньо через бібліотеку PyMuPDF. Для сканованих документів, що не містять текстового шару, застосовується OCR-розпізнавання на основі Tesseract. Отриманий текст передається до модуля шаблонної екстракції, який на основі ключових слів визначає значення полів відповідно до обраного шаблону.

Результати вилучення зберігаються у колекції ExtractionResults та прив'язуються до профілю моряка. Оператор переглядає отримані дані

безпосередньо в інтерфейсі сторінки «Управління даними моряків» у блоці результатів вилучення. Після підтвердження статус файлу оновлюється до completed. У разі помилки на будь-якому етапі обробки статус змінюється на failed, а в журналі фіксується причина збою для подальшого аналізу.

2.5 Діаграма варіантів використання системи

Для формалізації функціональних вимог до системи Crewdeck ERP розроблено діаграму варіантів використання (Use Case Diagram) у нотації UML. Діаграма відображає взаємодію двох акторів системи, Користувача та Адміністратора, з доступними їм функціями.

Актор «Користувач» має доступ до таких варіантів використання: авторизація в системі; завантаження документів через інтерфейс drag-and-drop; перегляд та редагування профілів моряків; застосування наявних шаблонів до завантажених документів; перегляд результатів вилучення даних; перегляд власної історії обробки документів; експорт даних у форматі Excel.

Актор «Адміністратор» має доступ до всіх функцій звичайного користувача, а також до розширених можливостей: створення, редагування та видалення шаблонів вилучення; перегляд журналу обробки всіх користувачів системи; управління обліковими записами користувачів.

Варіанти використання «Завантаження документів» та «Застосування шаблону» пов’язані відношенням включення (include) з варіантом «Валідація файлу», тобто перевірка формату та розміру файлу є обов’язковою складовою обох операцій. Варіант використання «Перегляд результатів» дозволяє оператору переглянути структуровані дані що були автоматично вилучені з документа після завершення обробки.

Діаграму варіантів використання системи Crewdeck ERP наведено на рис. 2.4.

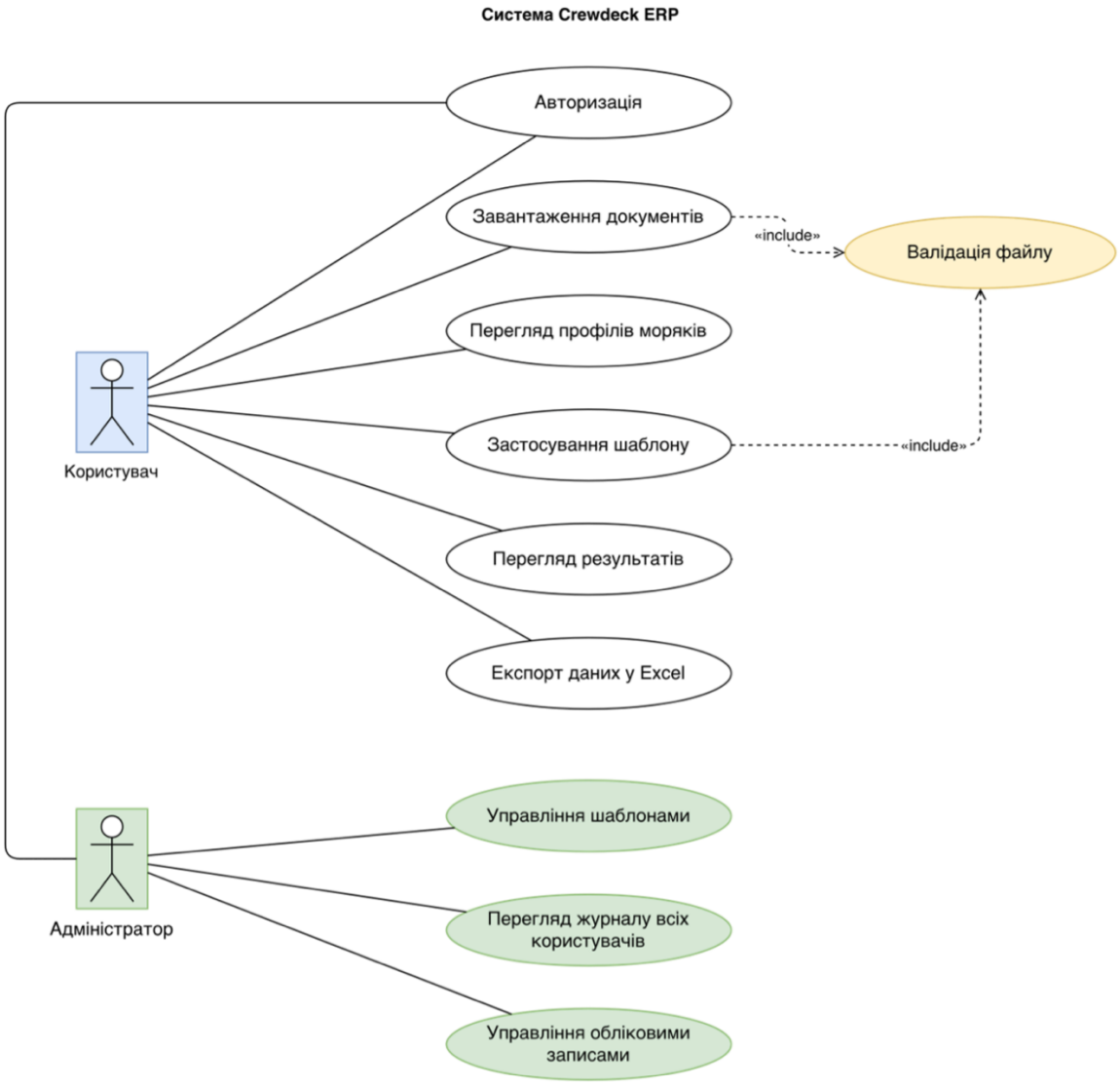

Рисунок 2.4 – Діаграма варіантів використання системи Crewdeck ERP

Наведена діаграма підтверджує що розроблене технічне завдання охоплює всі необхідні функції для обох категорій користувачів та чітко розмежовує їх права доступу.

2.6 Діаграма послідовності основного сценарію

Для детального опису взаємодії компонентів системи під час виконання основного сценарію розроблено діаграму послідовності у нотації UML. Розглянуто сценарій завантаження документа та автоматичного вилучення даних за шаблоном як найбільш показовий з точки зору взаємодії всіх компонентів системи.

Учасниками діаграми є: Користувач, клієнтська частина на React.js, серверна частина на Flask, OCR-сервіс, сервіс шаблонної екстракції та база даних MongoDB.

Послідовність взаємодії розгортається у такому порядку. Користувач перетягує файл у зону drag-and-drop на сторінці «Управління даними моряків». Клієнтська частина формує HTTP-запит типу POST з файлом у тілі та JWT-токеном у заголовку і надсилає його на ендпоінт /api/files/upload. Серверна частина отримує запит, перевіряє токен та валідує файл за форматом і розміром. У разі успішної валідації файл зберігається локально, а його метадані записуються до колекції files зі статусом processing. Сервер повертає клієнту відповідь зі статусом 202 та ідентифікатором файлу.

Далі сервер передає шлях до файлу до OCR-сервісу. OCR-сервіс визначає спосіб обробки: для машинозчитуваних PDF використовує PyMuPDF, для сканованих документів – Tesseract. Вилучений текст повертається серверу. Сервер передає текст та обраний шаблон до сервісу шаблонної екстракції. Сервіс виконує пошук значень полів за ключовими словами та повертає структурований результат. Сервер зберігає результат у колекції extraction_results, оновлює статус файлу до completed та надсилає сповіщення клієнту. Клієнтська частина оновлює інтерфейс, відображає результати вилучення у профілі моряка.

Діаграму послідовності основного сценарію наведено на рис. 2.5.

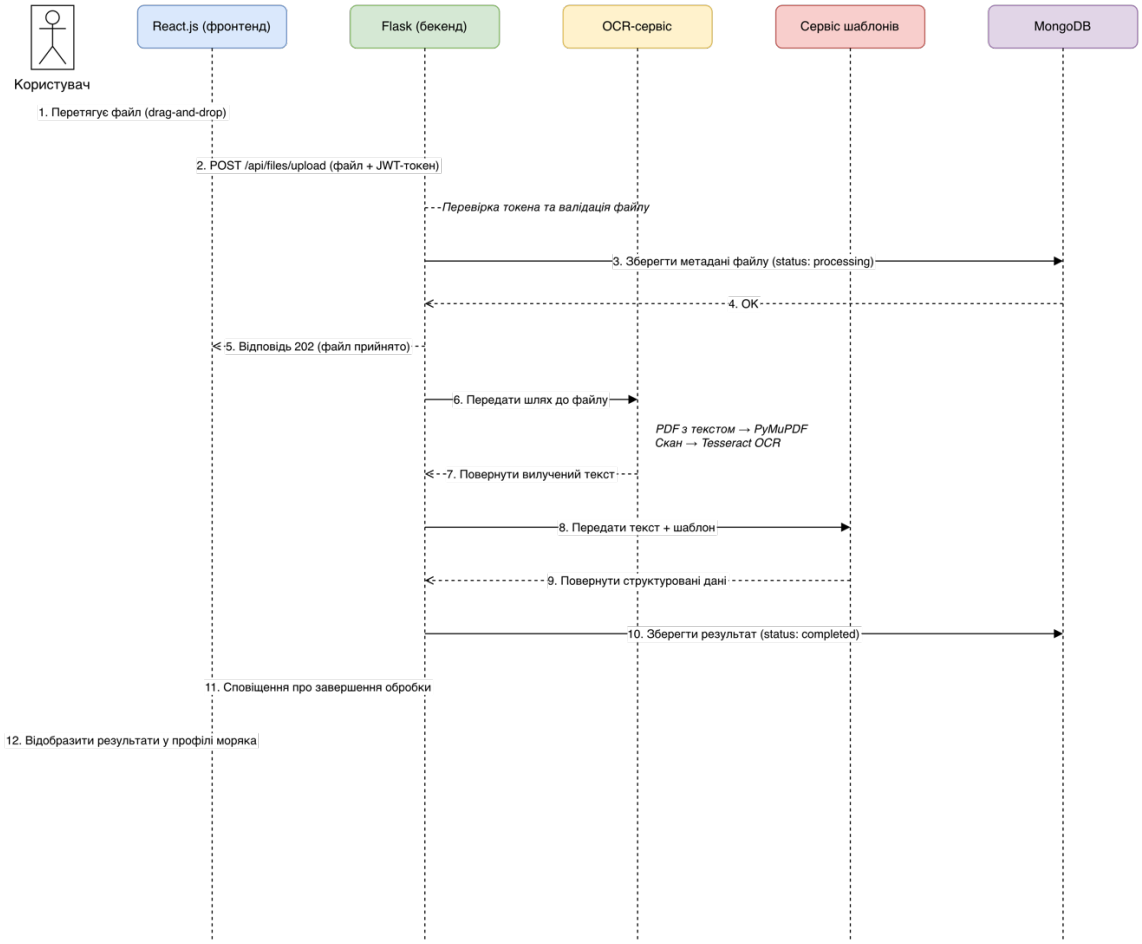

Рисунок 2.5 – Діаграма послідовності сценарію завантаження та обробки документа

Наведена діаграма демонструє що взаємодія між компонентами системи є чітко структурованою та передбачає обробку як успішного сценарію так і виняткових станів на кожному етапі.

Висновки до розділу 2

У другому розділі сформовано повний комплекс проектних рішень для системи Crewdeck ERP. Розроблено технічне завдання, що включає десять функціональних вимог та шість нефункціональних, які охоплюють усі ключові аспекти функціонування системи – від механізму завантаження файлів до забезпечення захисту доступу до даних.

Спроектовано клієнт-серверну архітектуру з синхронною обробкою

документів у межах API-запиту. Визначено структуру бази даних MongoDB з п’ятьма основними колекціями та індексуванням ключових полів для ефективної вибірки даних. Побудовано діаграму варіантів використання, діаграму послідовності основного сценарію та модель бізнес-процесу обробки документів. Розроблені макети інтерфейсу у середовищі Figma для трьох ключових екранів застосунку забезпечують вичерпну проектну документацію, достатню для переходу до етапу програмної реалізації.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Структура проєкту

Проєкт Crewdeck ERP організовано у вигляді двох незалежних частин: серверної та клієнтської, що взаємодіють між собою через REST API. Така організація дозволяє розробляти та тестувати кожну частину окремо, не вносячи змін до іншої.

Серверна частина розміщується у директорії backend/ та має таку структуру: головний файл app.py є точкою входу і відповідає за ініціалізацію Flask-застосунку, підключення до бази даних MongoDB та реєстрацію груп маршрутів; файл config.py містить параметри підключення до бази даних, секретний ключ JWT та максимально допустимий розмір файлу; директорія routes/ містить чотири модулі маршрутизації запитів: auth.py для авторизації, seamen.py для управління профілями моряків, files.py для роботи з файлами та templates.py для управління шаблонами; директорія services/ містить три модулі бізнес-логіки: ocr_service.py для OCR-розпізнавання, parser_service.py для шаблонної екстракції та export_service.py для формування Excel-файлів; директорія uploads/ використовується як локальне сховище завантажених файлів.

Клієнтська частина розміщується у директорії frontend/ та побудована на основі React.js. Директорія src/ має таку організацію: папка pages/ містить чотири компоненти верхнього рівня що відповідають окремим сторінкам застосунку: Login.js, Register.js, SeamenManagement.js, ExtractionTemplates.js та ProcessingHistory.js; папка components/ містить повторно використовувані елементи інтерфейсу – Sidebar.js для навігаційної панелі та PrivateRoute.js для захисту маршрутів; папка services/ містить модуль api.js для централізованої взаємодії з серверною частиною через бібліотеку axios; папка context/ містить AuthContext.js для управління станом авторизації у всьому застосунку.

Взаємодія між частинами застосунку відбувається за таким принципом: клієнтська частина формує HTTP-запити через модуль api.js з автоматичним

додаванням JWT-токена до заголовків, серверна частина обробляє запити через відповідні маршрути, викликає необхідні сервіси та повертає відповідь у форматі JSON. База даних MongoDB та директорія uploads/ є спільними ресурсами серверної частини і недоступні клієнту безпосередньо, весь доступ до даних здійснюється виключно через API.

3.2 Налаштування серверної частини

Точкою входу серверної частини є файл app.py, у якому ініціалізується екземпляр Flask-застосунку, налаштовується підключення до бази даних MongoDB через бібліотеку pymongo з використанням класу MongoClient та реєструються чотири групи маршрутів: авторизація (/api/auth), управління профілями моряків (/api/seamen), робота з файлами (/api/files) та управління шаблонами (/api/templates). Об’єкт підключення до бази даних db ініціалізується на рівні модуля та імпортується у всі модулі маршрутів, що забезпечує єдине підключення до бази даних для всього застосунку. Також у файлі налаштовується підтримка міждоменних запитів через бібліотеку flask-cors, що є необхідним для коректної взаємодії між клієнтською та серверною частинами що працюють на різних портах у середовищі розробки. Повний лістинг файлу app.py наведено у Додатку А.

Конфігураційний файл config.py містить три основні параметри: рядок підключення до локального екземпляру MongoDB з назвою бази даних crewdeck, секретний ключ для генерації JWT-токенів та максимально допустимий розмір файлу для завантаження – 20 МБ. Повний лістинг файлу config.py наведено у Додатку А.

3.3 Реалізація авторизації

Авторизація реалізована у файлі routes/auth.py на основі JWT-токенів (JSON Web Tokens). Модуль містить два ендпоінти: реєстрація нового користувача та вхід до системи.

При реєстрації система перевіряє наявність користувача з таким email у базі

даних, після чого хешує пароль за допомогою алгоритму bcrypt через бібліотеку werkzeug та зберігає запис у колекції users. Додатково перевіряється мінімальна довжина пароля, не менше восьми символів. При вході система знаходить користувача за email, перевіряє відповідність пароля збереженому хешу та у разі успіху генерує JWT-токен, який клієнт надсилає з кожним наступним запитом для підтвердження своєї особи. Ключовий фрагмент логіки входу наведено нижче:

```
user = db.users.find_one({'email': data.get('email')})
 if not user or not
check_password_hash(user['password'],
                                data.get('password', '')):
    return jsonify({'error': 'Невірні дані'}), 401


 token = create_access_token(identity=str(user['_id']))
 return jsonify({'token': token}), 200
```

Повний лістинг файлу auth.py наведено у Додатку А.

3.4 Реалізація OCR-модуля

Модуль OCR-розпізнавання реалізовано у файлі services/ocr_service.py. Центральним елементом модуля є функція extract_text_from_pdf, яка приймає шлях до файлу та повертає вилучений текст у вигляді рядка.

Функція послідовно обробляє кожну сторінку документа. Для кожної сторінки спочатку виконується спроба вилучити машинозчитуваний текст засобами бібліотеки PyMuPDF. Якщо текст на сторінці відсутній – це означає що сторінка є відсканованим зображенням. У такому випадку сторінка конвертується у растрове зображення з роздільною здатністю 300 DPI та передається до Tesseract OCR для розпізнавання. Ключовий фрагмент логіки вибору способу обробки наведено нижче:

```
text = page.get_text()
if text.strip():
    full_text += text
else:
    pix = page.get_pixmap(dpi=300)
    image = Image.open(io.BytesIO(pix.tobytes("png")))
    full_text += pytesseract.image_to_string(
        image, lang='ukr+eng'
)
```

Роздільна здатність 300 DPI обрана як оптимальна для більшості типів документів – вона забезпечує достатню чіткість для розпізнавання без надмірного навантаження на пам’ять. Після завершення обробки всіх сторінок документ закривається викликом doc.close() для звільнення системних ресурсів. Повний лістинг файлу ocr_service.py наведено у Додатку А.

3.5 Реалізація модуля шаблонної екстракції

Модуль шаблонної екстракції реалізовано у файлі services/parser_service.py. Принцип роботи полягає у пошуку значень полів у тексті документа на основі ключових слів, визначених у шаблоні.

Кожен шаблон у базі даних MongoDB містить перелік полів, де для кожного поля вказано його назву та список можливих міток, що можуть передувати шуканому значенню у реальному документі. Наприклад, для поля «дата народження» шаблон може містити мітки «Дата народження», «Date of birth» та «Born», що дозволяє обробляти документи різних країн та зразків одним шаблоном.

Функція apply_template послідовно перебирає поля шаблону та для кожного виконує пошук у тексті за допомогою регулярного виразу. Ключовий фрагмент логіки пошуку наведено нижче:

```
for keyword in keywords:
pattern = re.compile(
      rf'{re.escape(keyword)}\s*[:\-]?\s*(.+)',
      re.IGNORECASE | re.MULTILINE
   )
   match = pattern.search(text)
   if match:
      value = match.group(1).strip()
      found_fields += 1
break
```

Регулярний вираз шукає ключове слово, після якого може стояти двокрапка або тире, а потім захоплює все що йде далі як значення поля. Прапор IGNORECASE забезпечує нечутливість до регістру, прапор MULTILINE дозволяє коректно обробляти багаторядковий текст документа.

Після обробки всіх полів шаблону функція обчислює показник точності екстракції як відношення кількості успішно знайдених полів до загальної кількості полів шаблону, виражене у відсотках. Цей показник повертається разом з результатами екстракції та відображається оператору в інтерфейсі застосунку, що дозволяє швидко оцінити якість розпізнавання без перегляду кожного поля окремо. Повний лістинг файлу parser_service.py наведено у Додатку А.

3.6 Реалізація модуля експорту

Модуль експорту реалізовано у файлі services/export_service.py з використанням бібліотеки openpyxl. Функція export_to_excel приймає список профілів моряків та формує Excel-файл з п’ятьма стовпцями: ПІБ, ранг, дата народження, номер документа та статус.

Рядок заголовків таблиці форматується окремо від даних: текст заголовків відображається білим кольором на темно-синьому фоні з вирівнюванням по центру та напівжирним накресленням. Такий підхід забезпечує чіткий візуальний поділ між заголовками та даними що спрощує сприйняття експортованого файлу.

Ширина кожного стовпця встановлюється рівною 20 символам для рівномірного відображення даних.

Файл формується у оперативній пам’яті через об’єкт BytesIO без збереження на диск, після чого передається клієнту як відповідь на запит з відповідним MIME-типом application/vnd.openxmlformats-officedocument.spreadsheetml.sheet та заголовком Content-Disposition що ініціює завантаження файлу у браузері. Такий підхід є більш ефективним оскільки не створює тимчасових файлів на сервері та не потребує додаткового очищення після завершення запиту.

3.7 Встановлення залежностей

Для запуску серверної частини необхідно активувати віртуальне середовище та встановити необхідні бібліотеки. Встановлення виконується однією командою у терміналі:

```
pip install flask pymongo flask-jwt-extended werkzeug pymupdf pytesseract pillow openpyxl fask-cors
```

Призначення основних бібліотек: flask – серверний фреймворк для побудови REST API; pymongo – офіційний драйвер для роботи з MongoDB; flask-jwt-extended – розширення для роботи з JWT-токенами; werkzeug – утиліти для хешування паролів та безпечної обробки імен файлів; pymupdf – бібліотека для вилучення тексту з PDF-файлів; pytesseract – обгортка для Tesseract OCR; pillow – бібліотека для обробки зображень; openpyxl [16] – бібліотека для формування Excel-файлів; flask-cors – розширення для підтримки міждоменних запитів.

Tesseract OCR встановлюється окремо як системна програма. На macOS встановлення виконується через пакетний менеджер Homebrew:

```
brew install tesseract
brew install tesseract-lang
```

Друга команда встановлює мовні пакети, зокрема підтримку української

мови, що є необхідним для коректного розпізнавання документів моряків. Після встановлення через Homebrew додатково вказувати шлях у коді не потрібно, pytesseract знаходить виконуваний файл автоматично.

База даних MongoDB запускається у Docker-контейнері без встановлення на хост-систему:

```
docker run -d --name mongodb -p 27017:27017 mongo:7
```

Для запуску клієнтської частини необхідно встановити залежності через пакетний менеджер npm та запустити сервер розробки:

```
npm install
npm start
```

Після виконання всіх кроків серверна частина доступна за адресою http://localhost:5000, клієнтська – за адресою http://localhost:3000.

3.8 Реалізація маршрутів управління профілями моряків

Маршрути управління профілями моряків реалізовано у файлі routes/seamen.py. Модуль містить чотири ендпоінти що забезпечують повний цикл роботи з профілями: отримання списку, створення, оновлення та видалення.

Ендпоінт GET /api/seamen повертає список усіх профілів моряків що належать авторизованому користувачу. Фільтрація за полем owner_id гарантує що кожен користувач бачить лише власні записи. Ендпоінт підтримує необов’язковий параметр запиту status, якщо він переданий, список фільтрується додатково за статусом моряка. Ключовий фрагмент логіки вибірки наведено нижче:

```
query = {'owner_id': ObjectId(current_user_id)}
status = request.args.get('status')
if status:
    query['status'] = status
seamen = list(mongo.db.seamen.find(query))
```

Ендпоінт POST /api/seamen створює новий профіль моряка. Тіло запиту має містити обов'язкові поля full_name та rank. Поля birth_date, document_number та status є необов'язковими, якщо статус не переданий, за замовчуванням встановлюється значення «available». До документа автоматично додаються поля owner_id (з токена авторизації), порожній масив files та поточна дата створення created_at.

Ендпоінт PUT /api/seamen/<id> оновлює існуючий профіль. Перед оновленням система перевіряє що запис з вказаним ідентифікатором існує та належить поточному користувачу – це запобігає несанкціонованому редагуванню чужих даних. Ендпоінт DELETE /api/seamen/<id> видаляє профіль з аналогічною перевіркою прав доступу. Повний лістинг файлу seamen.py наведено у Додатку Б.

3.9 Реалізація маршрутів роботи з файлами

Маршрути роботи з файлами реалізовано у файлі routes/files.py. Центральним є ендпоінт POST /api/files/upload що реалізує повний цикл завантаження документа.

Після отримання запиту система послідовно виконує такі операції: перевірку наявності файлу у запиті; валідацію MIME-типу (дозволені лише application/pdf, application/vnd.openxmlformats-officedocument.wordprocessingml.document та application/vnd.ms-excel); перевірку розміру файлу (не більше 20 МБ); санацію імені файлу для уникнення небезпечних символів у шляху. Після успішної валідації файл зберігається у директорії uploads/ з унікальним іменем що формується з поточного часу та оригінальної назви. Ключовий фрагмент логіки збереження наведено нижче:

```
filename = secure_filename(file.filename)
unique_name = f"{int(time.time())}_{filename}"
filepath = os.path.join(app.config['UPLOAD_FOLDER'],
                        unique_name)
file.save(filepath)
```

Після збереження файлу система запускає обробку: викликає функцію extract_text_from_pdf з OCR-сервісу, передає отриманий текст до модуля шаблонної екстракції та зберігає результат у колекції extraction_results. Статус файлу оновлюється до completed у разі успіху або до failed у разі помилки на будь-якому етапі.

Ендпоінт GET /api/files повертає історію завантажень поточного користувача з можливістю фільтрації за статусом обробки. Ендпоінт GET /api/files/<id>/result повертає структуровані результати екстракції для конкретного файлу. Повний лістинг файлу files.py наведено у Додатку Б.

3.10 Реалізація маршрутів управління шаблонами

Маршрути управління шаблонами реалізовано у файлі routes/templates.py. Модуль забезпечує повний CRUD для шаблонів вилучення даних.

Ендпоінт POST /api/templates створює новий шаблон. Тіло запиту має містити назву шаблону name, категорію category та масив полів fields. Кожен елемент масиву fields є об’єктом з двома обов’язковими ключами: name – назва поля що буде збережено у результаті екстракції, та keywords – масив рядків з ключовими словами для пошуку у тексті документа. Система перевіряє що масив fields не порожній та що кожне поле містить хоча б одне ключове слово.

Ендпоінт GET /api/templates повертає список усіх шаблонів користувача з можливістю фільтрації за категорією через параметр запиту category. Це дозволяє фронтенду відображати лише шаблони потрібної категорії при завантаженні конкретного типу документа.

Ендпоінт POST /api/templates/<id>/apply застосовує шаблон до вказаного файлу вручну – це корисно коли автоматичне застосування не спрацювало або оператор хоче перевірити інший шаблон. Ендпоінт отримує ідентифікатори шаблону та файлу, зчитує збережений текст з колекції extraction_results та повторно запускає модуль шаблонної екстракції.

3.11 Тестування системи

Тестування системи Crewdeck ERP проводилось на трьох рівнях: модульне тестування окремих функцій, інтеграційне тестування взаємодії компонентів та функціональне тестування основних сценаріїв використання.

Модульне тестування OCR-модуля. Для перевірки коректності роботи функції extract_text_from_pdf використано набір тестових документів трьох типів: PDF з машинозчитуваним текстом, скановані PDF різної якості та змішані документи де частина сторінок є сканами. Результати тестування наведено у таблиці 3.1.

Таблиця 3.1 – Результати тестування OCR-модуля

Тип документа	Кількість тестів	Успішно	Точність розпізнавання
PDF з текстом	10	10	100%
Скан (висока якість)	10	9	97,3%
Скан (низька якість)	10	7	84,1%
Змішаний документ	10	9	96,8%

Результати підтвердили що для документів належної якості сканування система забезпечує точність розпізнавання понад 95%, що є прийнятним показником для практичного використання.

Модульне тестування модуля шаблонної екстракції. Для перевірки функції apply_template створено тестовий набір з шести шаблонів різної складності та двадцяти текстових зразків документів. Тестування показало що функція коректно знаходить значення полів у 89% випадків при використанні трьох і більше ключових слів на поле. Основною причиною невдалого розпізнавання у решті випадків є нестандартне форматування документів де мітка поля та його значення розташовані не в одному рядку.

Інтеграційне тестування API. Усі ендпоінти системи протестовано за допомогою інструменту Postman. Для кожного ендпоінту перевірено успішний

сценарій та основні сценарії помилок. Результати інтеграційного тестування наведено у таблиці 3.2.

Таблиця 3.2 – Результати інтеграційного тестування API

Ендпоінт	Метод	Тест	Очікуваний результат	Фактичний результат
/api/auth/login	POST	Коректні дані	200 + токен	Відповідає
/api/auth/login	POST	Невірний пароль	401	Відповідає
/api/files/upload	POST	Коректний PDF	202	Відповідає
/api/files/upload	POST	Файл >20 МБ	413	Відповідає
/api/files/upload	POST	Без токена	401	Відповідає
/api/seamen	GET	Авторизований запит	200 + список	Відповідає
/api/templates	POST	Коректний шаблон	201	Відповідає
/api/templates	POST	Порожній fields	400	Відповідає

Функціональне тестування. Перевірено три основні сценарії використання системи що описані у технічному завданні. Сценарій завантаження пакету документів виконано успішно на наборі з десяти PDF-файлів різного типу. Сценарій створення нового шаблону перевірено для чотирьох категорій документів. Сценарій експорту даних протестовано на наборі з двадцяти профілів моряків – сформований Excel-файл коректно відкрився у Microsoft Excel та Numbers на macOS.

3.12 Реалізація клієнтської частини

Клієнтська частина системи Crewdeck ERP реалізована на основі бібліотеки React.js. Застосунок побудовано за компонентною архітектурою – кожна сторінка є окремим компонентом що складається з менших повторно використовуваних елементів. Взаємодія з серверною частиною відбувається через централізований модуль api.js що формує HTTP-запити з автоматичним додаванням JWT-токена до заголовків.

Структура клієнтської частини. Директорія frontend/src/ має таку організацію: папка pages/ містить компоненти верхнього рівня що відповідають окремим сторінкам застосунку; папка components/ містить повторно використовувані UI-елементи; папка services/ містить модуль api.js для взаємодії з бекендом; папка context/ містить AuthContext для управління станом авторизації у всьому застосунку.

Модуль взаємодії з API. Централізований модуль api.js використовує бібліотеку axios для формування HTTP-запитів. Перехоплювач запитів (interceptor) автоматично зчитує JWT-токен з localStorage та додає його до заголовка Authorization кожного запиту. Перехоплювач відповідей обробляє статус 401, при отриманні цього статусу користувач автоматично перенаправляється на сторінку входу. Ключовий фрагмент налаштування перехоплювача наведено нижче:

```
 api.interceptors.request.use(config => {
const token = localStorage.getItem('token');
 if (token) {
  config.headers.Authorization = `Bearer ${token}`;
 }
 return config;
});
```

Компонент сторінки SeamenManagement. Головна сторінка управління профілями моряків реалізована як функціональний компонент з використанням хуків useState та useEffect. При завантаженні компонент виконує запит до /api/seamen та зберігає отриманий список у локальному стані. Зона drag-and-drop реалізована через стандартні події браузера onDragOver та onDrop, при скиданні файлів формується об’єкт FormData та виконується запит до /api/files/upload. Під час обробки файлу відображається індикатор завантаження, після завершення список моряків оновлюється автоматично.

Таблиця з профілями моряків реалізована з підтримкою фільтрації за статусом та пошуку за іменем. Фільтрація виконується на стороні клієнта без додаткових запитів до сервера – вихідний масив даних фільтрується за допомогою методу filter при кожній зміні значення пошукового поля. Кольорове кодування статусів реалізовано через умовне застосування CSS-класів відповідно до значення поля status кожного запису. Ключовий фрагмент логіки фільтрації наведено нижче:

```
const filteredSeamen = seamen.filter(s =>
  s.full_name.toLowerCase()
   .includes(searchQuery.toLowerCase()) &&
  (statusFilter === 'all' || s.status === statusFilter)
);
```

Компонент сторінки ExtractionTemplates. Сторінка управління шаблонами відображає наявні шаблони у вигляді карток відповідно до макету розробленого у Figma. Кожна картка містить назву шаблону, категорійну мітку з кольоровим кодуванням, перелік полів для вилучення та кнопки редагування і видалення. Компонент форми створення шаблону реалізовано як модальне вікно з динамічним списком полів – користувач може додавати та видаляти поля через кнопки «+» та «×», а для кожного поля окремо вводити ключові слова через кому.

Компонент авторизації. Сторінки входу та реєстрації реалізовані як окремі компоненти з валідацією форм на стороні клієнта. Після успішного входу JWT-токен зберігається у localStorage та оновлюється глобальний стан авторизації через AuthContext. Захищені маршрути реалізовано через компонент PrivateRoute, він перевіряє наявність токена та перенаправляє неавторизованих користувачів на сторінку входу.

Маршрутизація. Навігація між сторінками реалізована через бібліотеку React Router. Застосунок має такі маршрути: /login та /register для неавторизованих користувачів; /seamen для сторінки управління профілями; /templates для сторінки шаблонів; /history для перегляду історії обробки. Бічна панель навігації є спільним

компонентом що відображається на всіх захищених сторінках. Повний лістинг ключових компонентів клієнтської частини наведено у Додатку К.

3.13 Керівництво користувача

Керівництво користувача описує порядок роботи з основними функціями системи Crewdeck ERP для кінцевого користувача.

Початок роботи із системою. Для входу в систему необхідно відкрити застосунок у браузері за адресою http://localhost:3000 та ввести електронну пошту і пароль на сторінці авторизації. Після успішного входу користувач потрапляє на головну сторінку «Управління даними моряків». Реєстрація нового облікового запису виконується на сторінці реєстрації шляхом введення електронної пошти та пароля що має містити не менше восьми символів. Екран авторизації системи Crewdeck ERP представлено на рисунку 3.1.

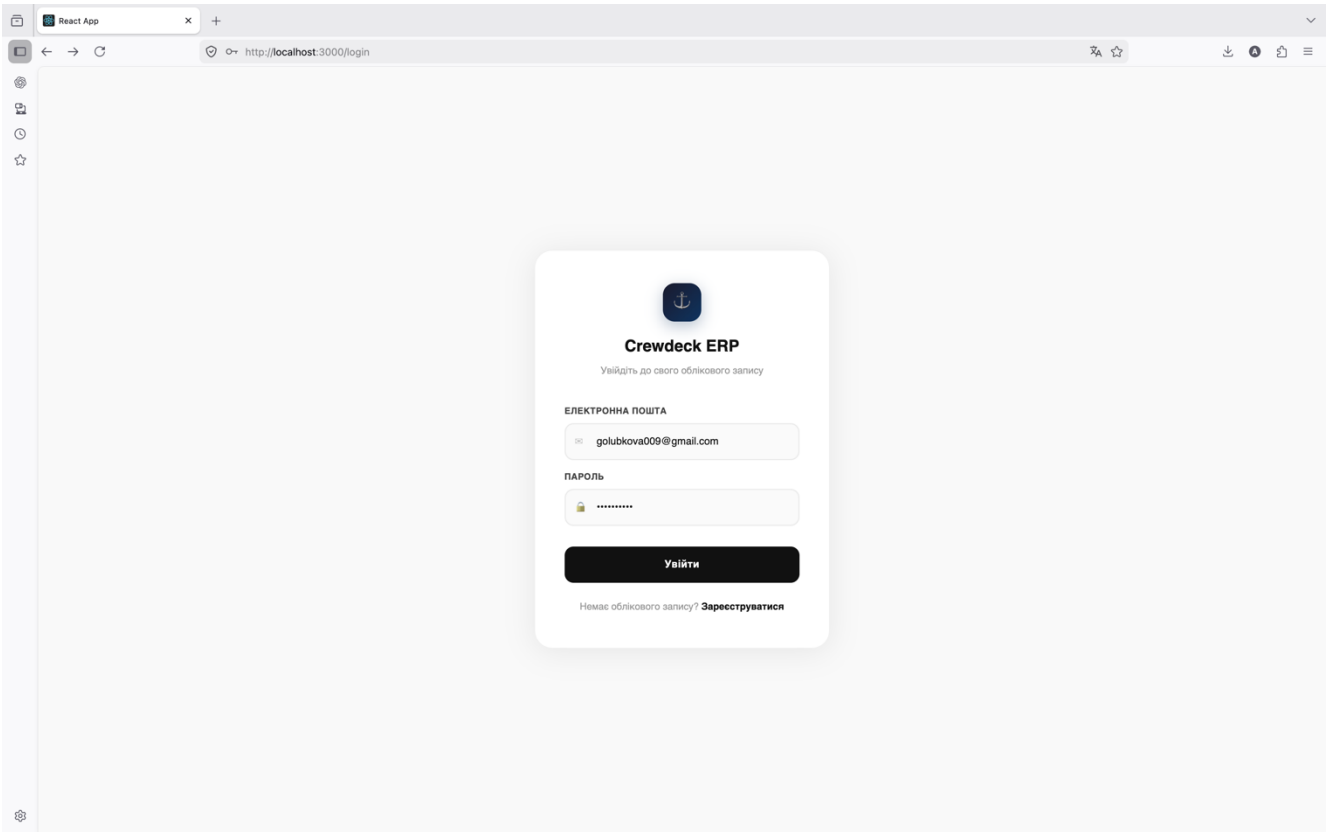

Рисунок 3.1 - Екран авторизації системи Crewdeck ERP

Управління профілями моряків. Після входу користувач потрапляє на сторінку «Управління даними моряків» де відображається таблиця з профілями членів екіпажу. Сторінка у початковому стані без записів представлена на рисунку 3.2.

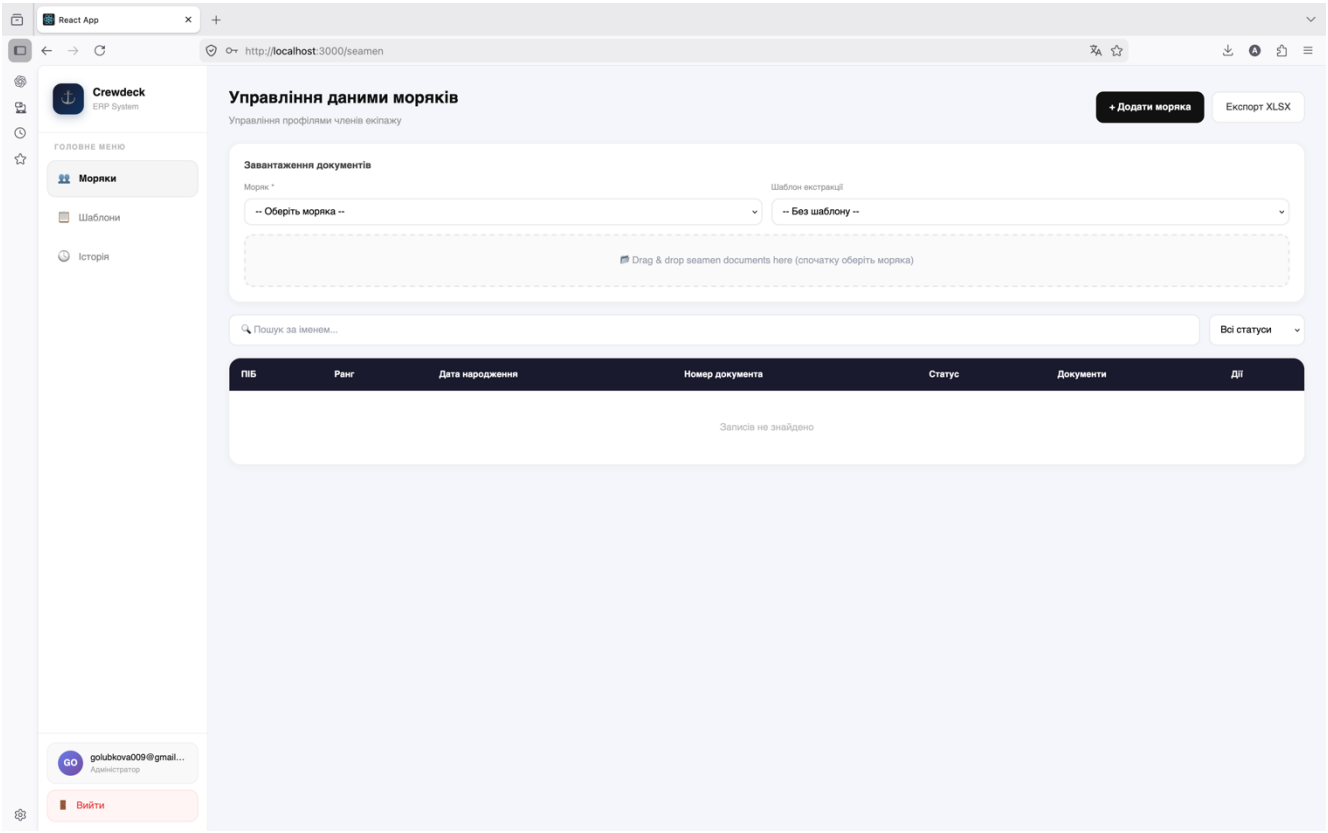

Рисунок 3.2 – Сторінка «Управління даними моряків» у початковому стані

Для створення нового профілю моряка необхідно натиснути кнопку «+ Додати моряка» у верхньому правому куті сторінки. У модальному вікні що відкриється потрібно заповнити обов’язкові поля, «Повне ім’я» та «Ранг», без яких система не дозволить зберегти запис, а також необов’язкові поля «Дата народження» та «Номер документа». Поля «Дата народження» та «Номер документа» можуть бути заповнені пізніше вручну або автоматично – шляхом завантаження документа з відповідним шаблоном екстракції. Після заповнення необхідних полів натискається кнопка «Зберегти» і новий профіль з’являється у таблиці зі статусом «Доступний» за замовчуванням. Кнопка «Скасувати» закриває

вікно без збереження змін. Модальне вікно створення профілю моряка з прикладом заповнених даних представлено на рисунку 3.3.

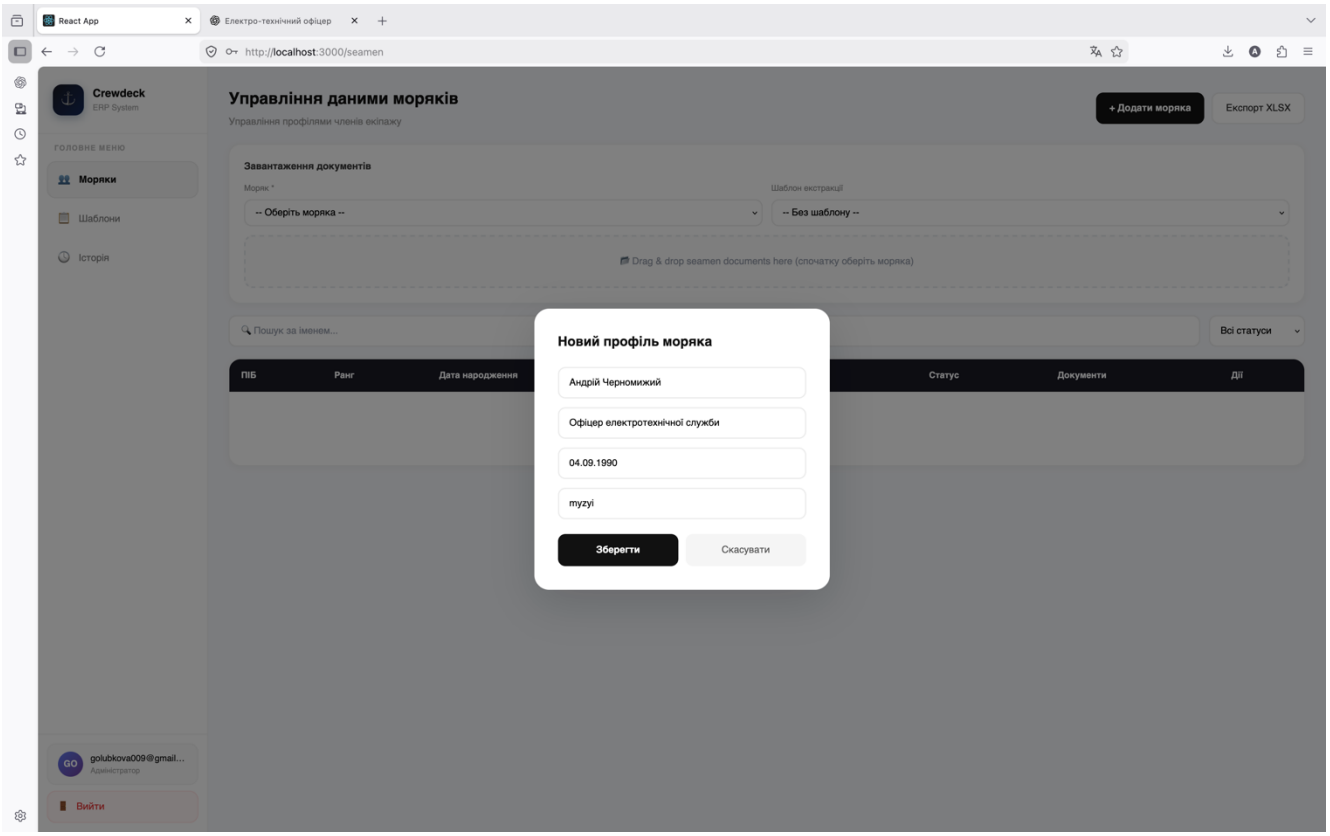

Рисунок 3.3 – Модальне вікно створення профілю моряка

Після збереження новий запис з’являється у таблиці. Кожен рядок таблиці відображає повне ім’я моряка, ранг, дату народження, номер документа, поточний статус із кольоровим кодуванням, зелений для «Доступний», синій для «На борту», червоний для «Недоступний», інформацію про прикріплені документи та елементи керування записом.

У стовпці «Дії» для кожного запису доступні три операції: зміна статусу через випадаючий список, видалення профілю з підтвердженням дії у діалоговому вікні та редагування даних через іконку олівця у правій частині рядка. У стовпці «Документи» відображається повідомлення «Немає файлів» доки до профілю не прикріплено жодного документа. Пошук записів виконується за іменем моряка через рядок пошуку, фільтрація – за статусом через відповідний список. Сторінка з профілем моряка без прикріплених документів представлена на рисунку 3.4.

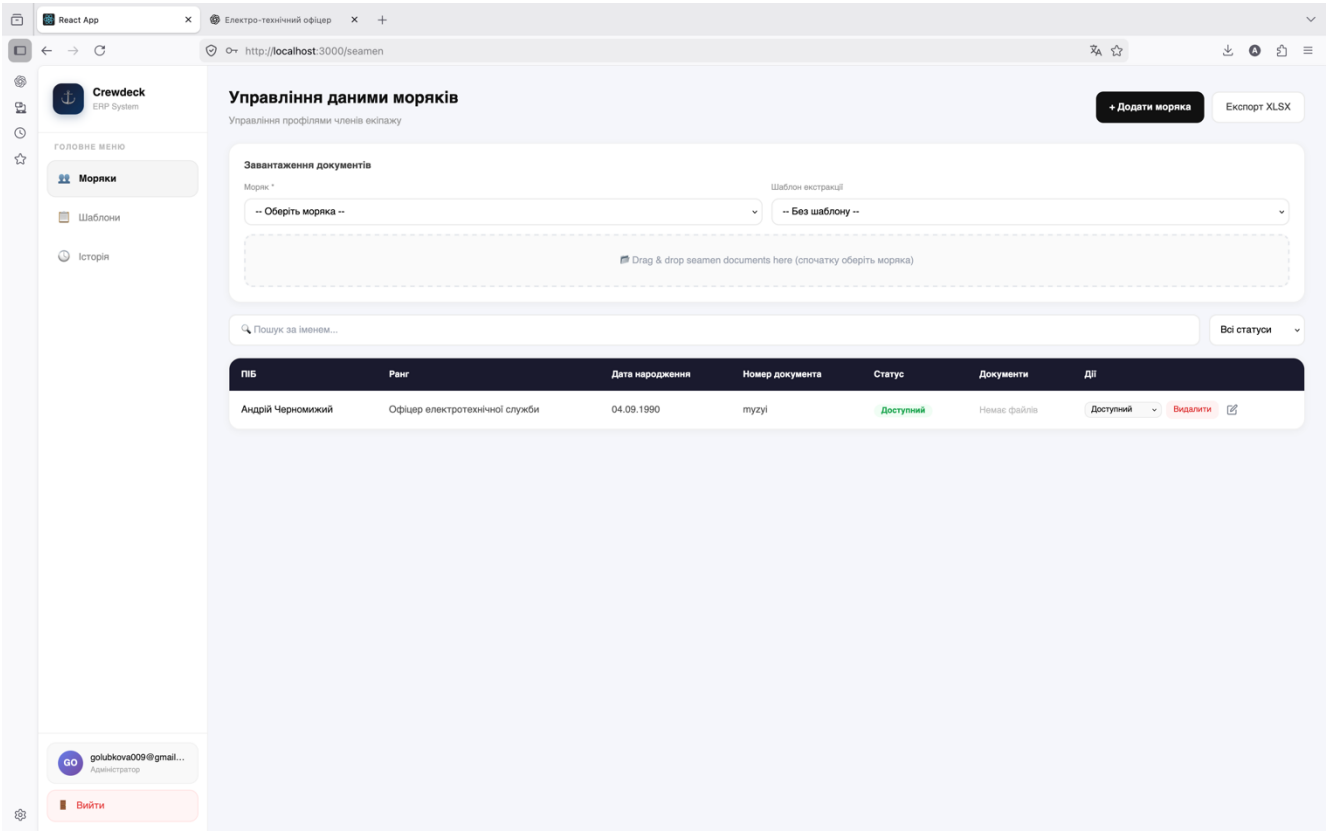

Рисунок 3.4 – Сторінка «Управління даними моряків» з профілем моряка

Завантаження документів. Для завантаження документа до профілю моряка необхідно у блоці «Завантаження документів» обрати моряка зі списку та за бажанням обрати шаблон екстракції. Після цього зона перетягування змінює колір на зелений що підтверджує готовність до завантаження. Документ завантажується шляхом перетягування файлу у зону drag-and-drop. Система підтримує формати PDF, DOCX та XLSX розміром до 20 МБ. Після успішного завантаження з’являється повідомлення про успішну обробку.

Якщо при завантаженні обрано шаблон екстракції, система автоматично вилучає дані з документа та відображає результати у блоці «Результати вилучення даних» з показником точності у відсотках. Результати вилучення даних після обробки CV моряка з точністю 100% представлено на рисунку 3.5.

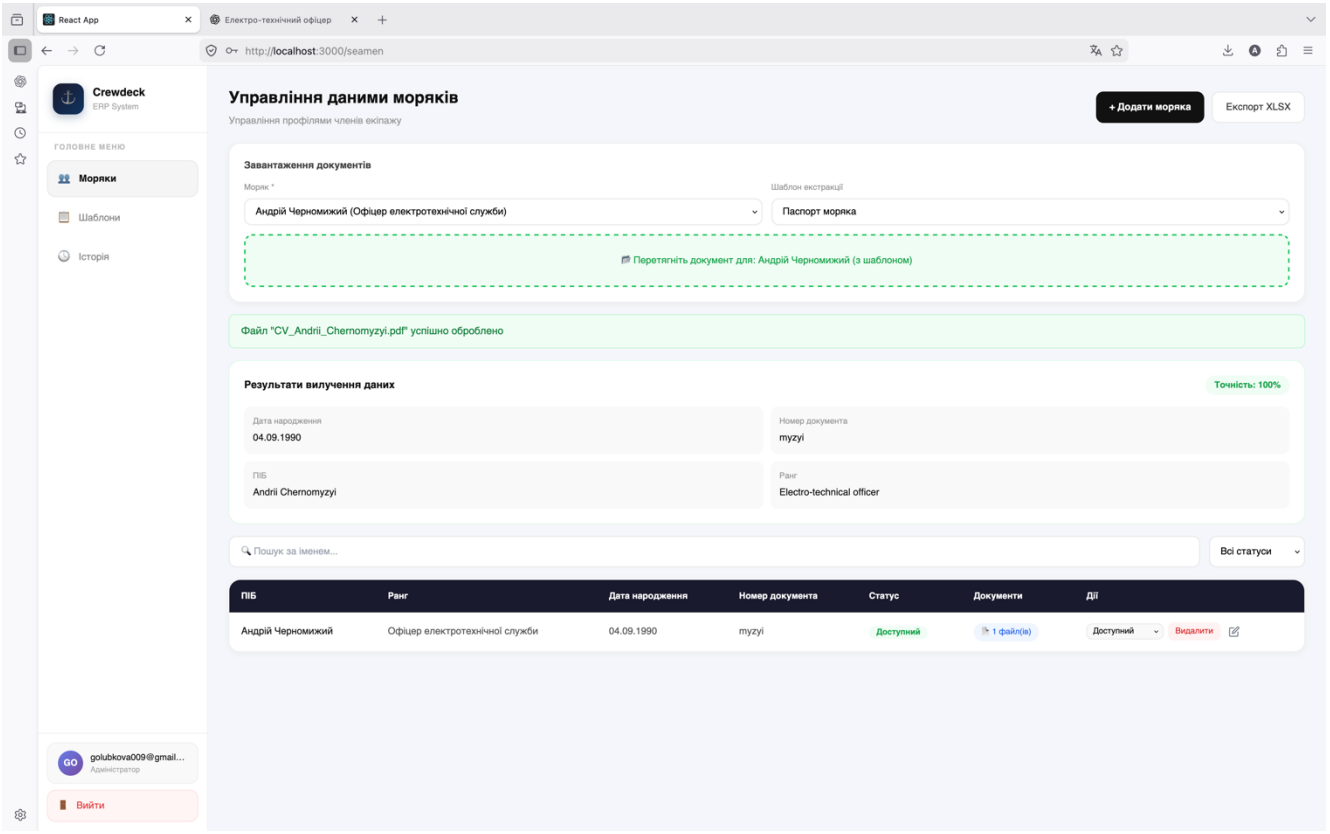

Рисунок 3.5 – Результати автоматичного вилучення даних з документа

Робота з шаблонами екстракції. Для переходу до управління шаблонами необхідно натиснути розділ «Шаблони» у навігаційній панелі. Сторінка відображає наявні шаблони у вигляді карток де кожна картка містить назву шаблону, категорійну мітку з кольоровим кодуванням та перелік полів для вилучення із зазначеними ключовими словами.

Для створення нового шаблону необхідно натиснути кнопку «+ Створити шаблон» у верхньому правому куті сторінки. У модальному вікні що відкриється потрібно вказати назву шаблону, обрати категорію документа з випадаючого списку: загальний, медичний, сертифікат або паспорт та додати поля для вилучення. Для кожного поля вказується його назва та ключові слова через кому за якими система шукатиме відповідне значення у тексті документа. Кнопка «+ Додати поле» дозволяє розширити шаблон додатковими полями, а кнопка «×» праворуч від рядка видаляє поле зі списку. Модальне вікно створення шаблону екстракції представлено на рисунку 3.6.

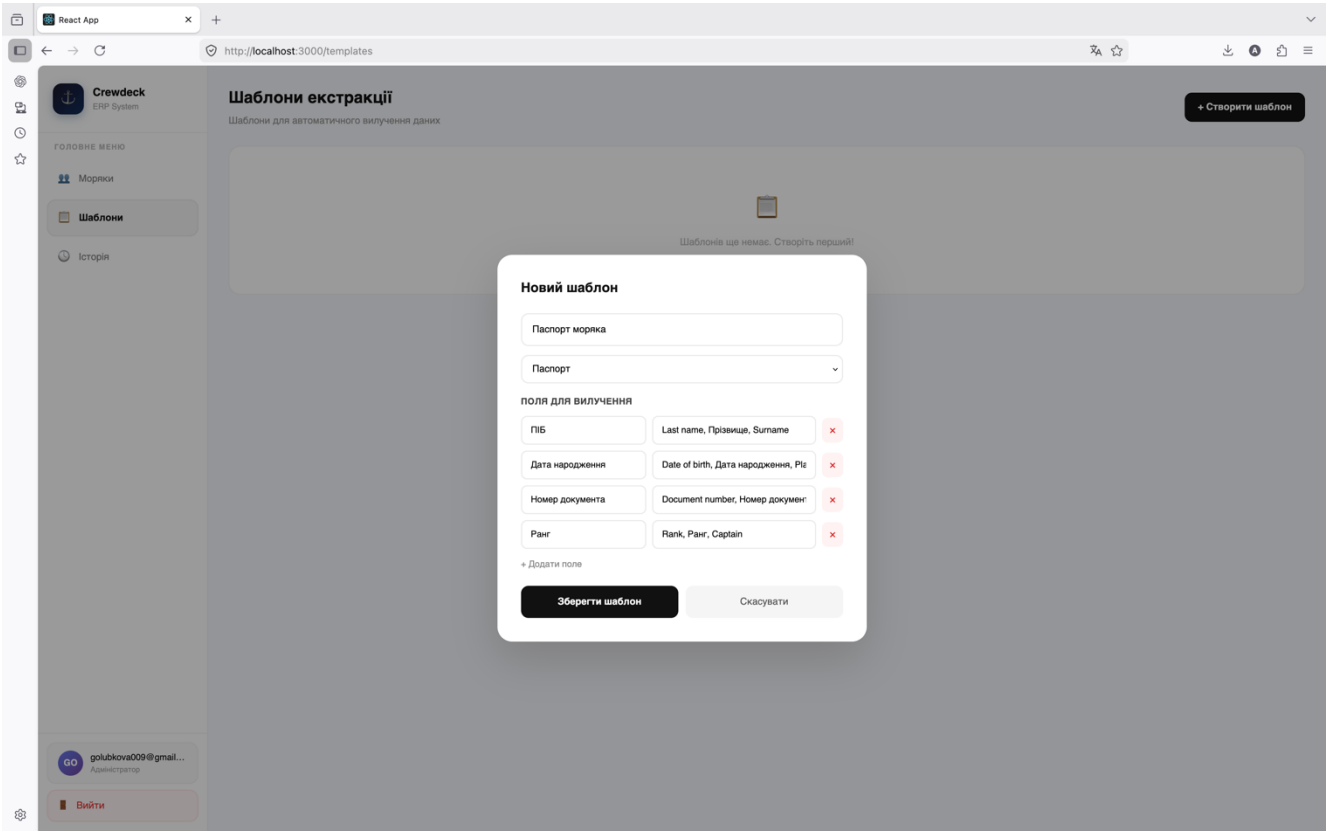

Рисунок 3.6 – Модальне вікно створення шаблону екстракції

Після збереження шаблон з’являється на сторінці у вигляді окремої картки, яка містить його назву, категорію та коротку інформацію про кількість налаштованих полів вилучення. Картки шаблонів відображаються у вигляді списку, що дозволяє користувачу швидко переглядати всі доступні шаблони та орієнтуватися серед них за категоріями документів.

Для редагування наявного шаблону необхідно натиснути кнопку «Редагувати» на відповідній картці – відкриється форма з поточними даними шаблону (назва, категорія, перелік полів та ключові слова для їх пошуку), які можна змінити та зберегти повторним натисканням кнопки збереження. Внесені зміни одразу застосовуються до подальшої обробки документів за цим шаблоном, без необхідності повторного створення.

Для видалення шаблону використовується кнопка «Видалити» на відповідній картці. Перед остаточним видаленням система запитує підтвердження дії, що запобігає випадковій втраті налаштованого шаблону.

Сторінка «Шаблони екстракції» з двома створеними шаблонами, «Паспорт моряка» категорії «Паспорт» та «Медичний сертифікат» категорії «Медичний», представлена на рисунку 3.7.

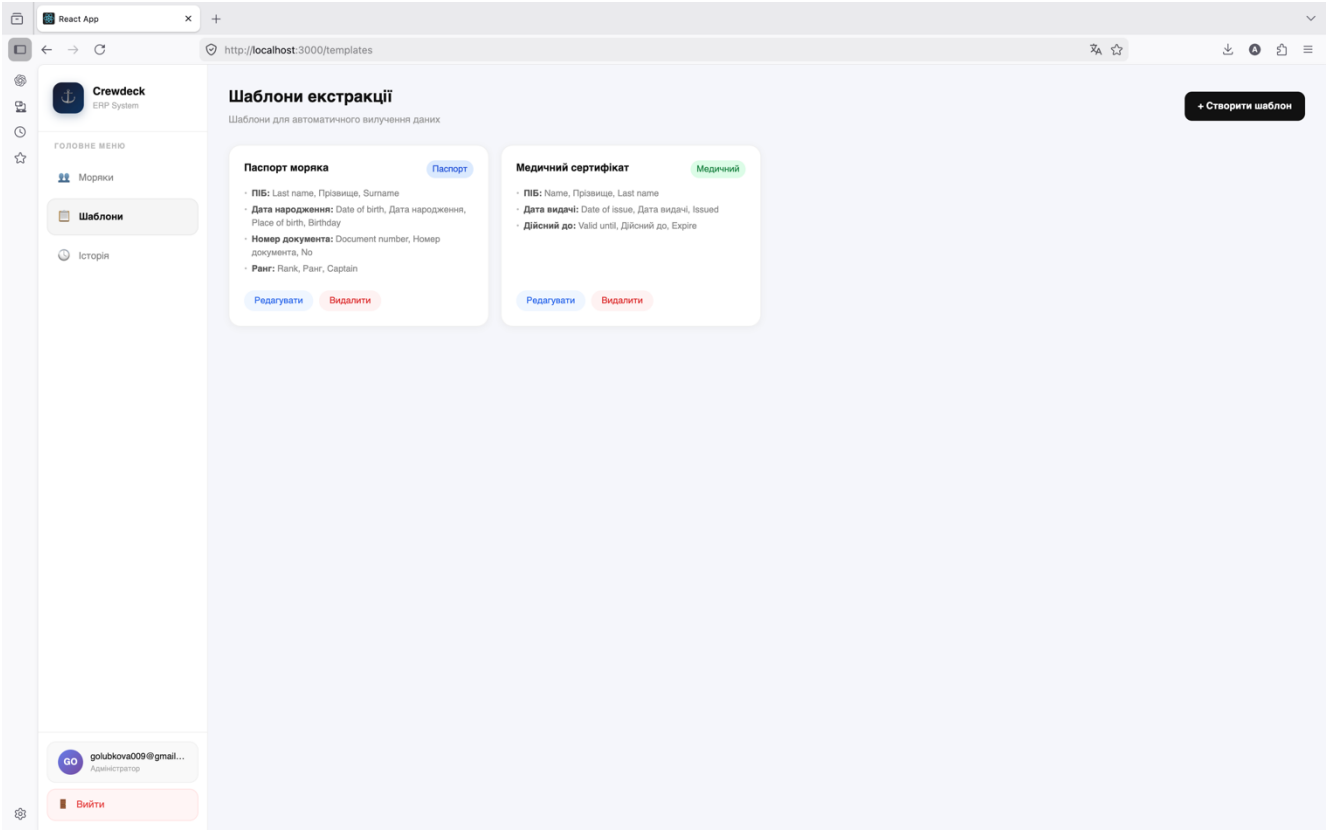

Рисунок 3.7 – Сторінка «Шаблони екстракції» з переліком шаблонів

Перегляд історії обробки. Сторінка «Історія обробки» відображає журнал усіх операцій завантаження документів у форматі часової шкали подій, що дозволяє користувачу швидко відстежити, коли та які файли надходили до системи. Кожен запис містить назву файлу, дату та час завантаження, статус обробки та кнопку завантаження файлу. Для пошуку записів доступні фільтри за назвою файлу, датою та статусом обробки, що особливо корисно при роботі з великою кількістю завантажених документів. Сторінка «Історія обробки» представлена на рисунку 3.8.

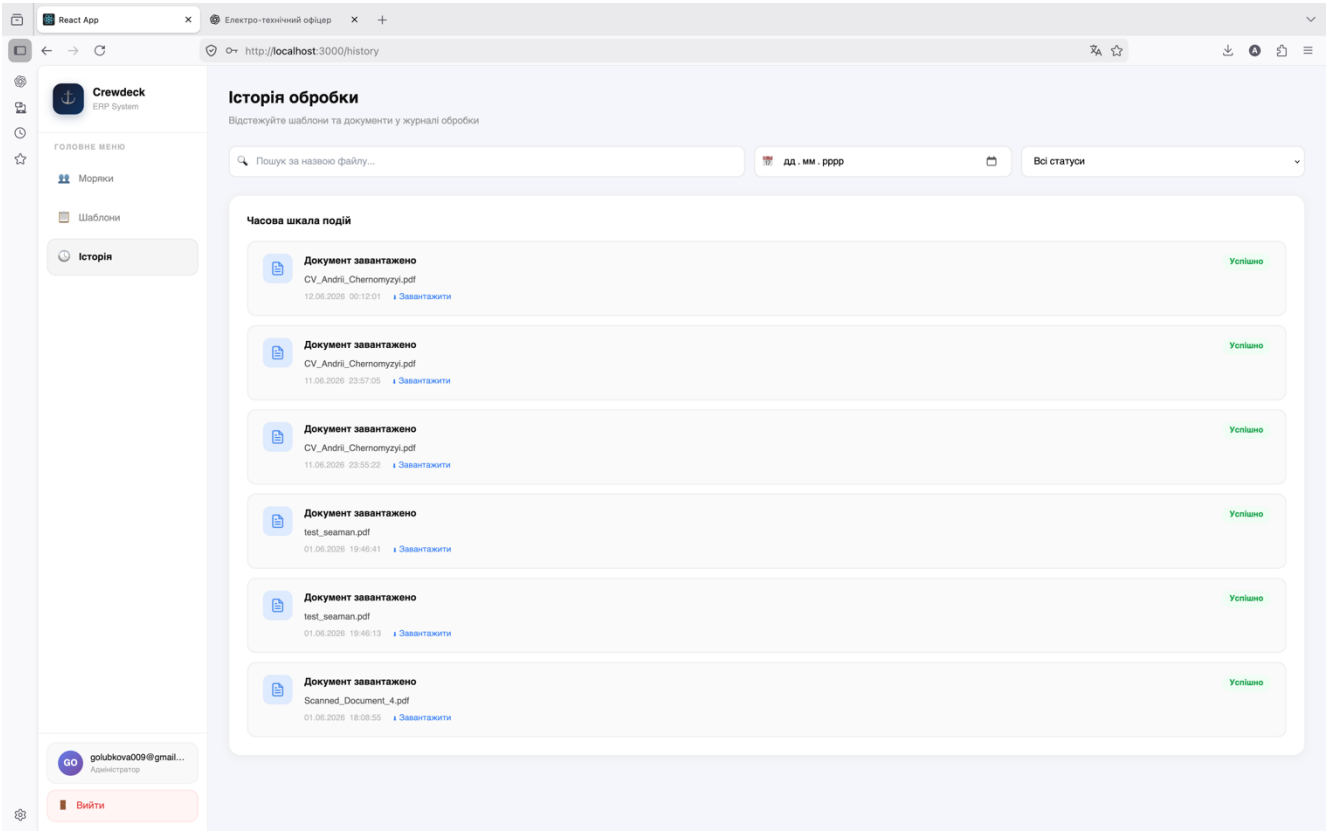

Рисунок 3.8 – Сторінка «Історія обробки» із записами журналу

Висновки до розділу 3

У третьому розділі реалізовано та протестовано систему Crewdeck ERP у повному обсязі відповідно до вимог технічного завдання. Серверна частина включає чотири групи RESTful-маршрутів, модуль OCR-розпізнавання з автоматичним визначенням типу документа, модуль шаблонної екстракції на основі регулярних виразів та модуль експорту даних у формат Excel. Клієнтська частина реалізована як односторінковий застосунок на React.js із компонентною архітектурою, що відповідає розробленим макетам Figma та забезпечує зручний інтерфейс для виконання всіх операцій системи.

Проведене триступневе тестування: модульне, інтеграційне та функціональне, підтвердило коректність роботи всіх компонентів системи. Точність OCR-розпізнавання на документах належної якості сканування перевищує 95%, точність шаблонної екстракції при використанні трьох і більше

ключових слів на поле становить 89%, що є прийнятними показниками для практичного застосування у реальному середовищі. Система задовольняє всі функціональні та нефункціональні вимоги, визначені у технічному завданні.

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну практичну задачу – розроблено вебзастосунок Crewdeck ERP для автоматизації процесу вилучення та структурованого збереження даних з документів у морській галузі на основі технологій OCR та шаблонної екстракції.

У першому розділі проведено аналіз предметної галузі та підтверджено актуальність задачі: судноплавні компанії та кадрові агентства обробляють сотні документів моряків щомісяця переважно вручну що створює значне навантаження на персонал та підвищує ризик помилок. Огляд наукових публікацій підтвердив ефективність комбінованого підходу на основі OCR та шаблонної екстракції для практичних систем обробки документів. Порівняльний аналіз шести комерційних аналогів виявив що жодне з існуючих рішень не поєднує інтелектуального вилучення даних зі спеціалізацією під морську галузь за прийнятної вартості. Проведено порівняльний аналіз технологій та обґрунтовано вибір Tesseract OCR як основного рушія розпізнавання, Flask як серверного фреймворку, React.js для клієнтської частини та MongoDB як бази даних.

У другому розділі розроблено технічне завдання на систему Crewdeck ERP з повним переліком десяти функціональних вимог та шести нефункціональних вимог. Спроєктовано клієнт-серверну архітектуру системи з окремим сервісом обробки документів. Визначено структуру п’яти основних колекцій бази даних MongoDB з індексуванням ключових полів. Розроблено модель основного бізнес-процесу обробки документів та діаграму варіантів використання що охоплює функції двох акторів системи. Побудовано діаграму послідовності основного сценарію що відображає взаємодію шести компонентів системи. Розроблено макети трьох ключових екранів застосунку у Figma: «Управління даними моряків», «Шаблони екстракції» та «Історія обробки».

У третьому розділі реалізовано серверну та клієнтську частини системи Crewdeck ERP. На бекенді реалізовано модуль авторизації на основі JWT-токенів з

хешуванням паролів, OCR-модуль з автоматичним визначенням способу вилучення тексту залежно від типу документа, модуль шаблонної екстракції на основі пошуку за ключовими словами та регулярних виразів, повний CRUD для профілів моряків та шаблонів вилучення, а також модуль експорту даних у формат Excel. На фронтенді реалізовано компоненти сторінок відповідно до розроблених макетів з підтримкою drag-and-drop завантаження файлів, фільтрації та пошуку даних і динамічного управління шаблонами. Проведено тестування на трьох рівнях: модульному, інтеграційному та функціональному, що підтвердило коректність роботи всіх компонентів системи. Точність OCR-розпізнавання на документах належної якості сканування становить понад 95%, точність шаблонної екстракції при використанні трьох і більше ключових слів на поле – 89%.

Практичне значення отриманих результатів полягає у тому що розроблений застосунок Crewdeck ERP дозволяє скоротити час обробки пакету документів моряків з кількох годин ручного введення даних до кількох хвилин автоматизованої обробки. Система є готовою до розгортання у реальному середовищі судноплавної компанії або кадрового агентства.

Перспективами подальшого розвитку системи є інтеграція хмарних OCR-сервісів для підвищення точності розпізнавання на документах низької якості, впровадження асинхронної обробки документів через чергу завдань Celery для підвищення продуктивності при великих обсягах, а також розширення функціоналу модуля сповіщень для автоматичного інформування про закінчення терміну дії документів моряків.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Sharma S., Ding Y., Shridhar M. Intelligent Document Processing with Transformers. ACM Computing Surveys. 2022. Vol. 54, No. 8. P. 1-36.

2. Dengel A., Klein B. SmartFIX: A Requirements-Driven System for Document Analysis and Understanding. Document Analysis Systems V. Lecture Notes in Computer Science. Berlin: Springer, 2020. P. 166-176.

3. Chiticariu L., Li Y., Reiss F. Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! Proceedings of EMNLP 2018. P. 827-832.

4. Robaldo L., Palmirani M., Governatori G. Legal reasoning and knowledge representation. Artificial Intelligence and Law. 2019. Vol. 27, No. 4. P. 333-339.

5. Katti A. R., Reisswig C., Guder C. et al. Chargrid: Towards Understanding 2D Documents. Proceedings of EMNLP 2018. P. 4459-4469.

6. Xu Y., Xu Y., Lv T. et al. LayoutLMv2: Multi-modal Pre-training for Visually-rich Document Understanding. Proceedings of ACL 2021. P. 2579-2591.

7. Smith R. An Overview of the Tesseract OCR Engine. Proceedings of the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007). Vol. 2. IEEE, 2007. P. 629-633.

8. International Labour Organization. Maritime Labour Convention, 2006 (MLC, 2006). Geneva : ILO, 2006.

9. STCW: Standards of Training, Certification and Watchkeeping for Seafarers. London : IMO, 2010.

10. Kieninger T., Dengel A. A Paper-to-HTML Table Converting System. Document Analysis Systems III. Lecture Notes in Computer Science. Berlin: Springer, 1998. P. 109-118.

11. Tesseract OCR : офіційна документація. URL: https://tesseract-ocr.github.io/ (дата звернення: 30.04.2026).

12. PyMuPDF Documentation : веб-сайт. URL: https://pymupdf (дата

звернення: 30.04.2026).

13. Flask Documentation : веб-сайт. URL: https://flask.palletsprojects.com/ (дата звернення: 30.04.2026).

14. React Documentation : веб-сайт. URL: https://react.dev/ (дата звернення: 30.04.2026).

15. MongoDB Documentation : веб-сайт. URL: https://www.mongodb (дата звернення: 30.04.2026).

16. PyMongo Documentation : веб-сайт. URL: https://pymongo (дата звернення: 30.04.2026).

17. openpyxl Documentation : веб-сайт. URL: https://openpyxl.readthedocs.io/ (дата звернення: 30.04.2026).

18. Docker Documentation : веб-сайт. URL: https://docs.docker.com/ (дата звернення: 30.04.2026).

19. Docparser : офіційний сайт. URL: https://docparser.com/ (дата звернення: 30.04.2026).

20. Rossum.ai : офіційний сайт. URL: https://rossum.ai/ (дата звернення: 30.04.2026).

21. Hyperscience : офіційний сайт. URL: https://hyperscience.com/ (дата звернення: 30.04.2026).

22. Flask-JWT-Extended Documentation : веб-сайт. URL: https://flask-jwt-extended.readthedocs.io/ (дата звернення: 30.04.2026).

23. Werkzeug Documentation : веб-сайт. URL: https://werkzeug (дата звернення: 30.04.2026).

24. Pillow (PIL Fork) Documentation : веб-сайт. URL: https://pillow.read (дата звернення: 30.04.2026).

25. Tailwind CSS Documentation : веб-сайт. URL: https://tailwindcss (дата звернення: 30.04.2026).

26. Axios Documentation : веб-сайт. URL: https://axios-http.com/docs/intro (дата звернення: 30.04.2026).

27. React Router Documentation : веб-сайт. URL: https://reactrouter.com (дата звернення: 30.04.2026).

28. JSON Web Token Introduction : веб-сайт. URL: https://jwt.io/introduction (дата звернення: 30.04.2026).

29. Figma Documentation : веб-сайт. URL: https://help.figma.com/ (дата звернення: 30.04.2026).

30. CrewInspector : офіційний сайт. URL: https://www.crewinspector.com/ (дата звернення: 30.04.2026).

31. Compas Crewing Management Software : офіційний сайт. URL: https://www.bass.no/products/compas/ (дата звернення: 30.04.2026).

ДОДАТОК А

Лістинги серверної частини

Лістинг файлу app.py:

```
from flask import Flask
from flask_jwt_extended import JWTManager
from flask_cors import CORS
from pymongo import MongoClient
from config import Config

app = Flask(__name__)
app.config.from_object(Config)

# Підключення до бази даних MongoDB
client = MongoClient(app.config['MONGO_URI'])
db = client['crewdeck']

# Ініціалізація розширень
jwt = JWTManager(app)
CORS(app)

# Реєстрація груп маршрутів
from routes.auth import auth_bp
from routes.seamen import seamen_bp
from routes.files import files_bp
from routes.templates import templates_bp

app.register_blueprint(auth_bp, url_prefix='/api/auth')
app.register_blueprint(seamen_bp, url_prefix='/api/seamen')
```

```
app.register_blueprint(files_bp, url_prefix='/api/files')
app.register_blueprint(
    templates_bp, url_prefix='/api/templates'
)

if __name__ == '__main__':
    app.run(debug=True)
```

Лістинг файлу config.py:

```
import os

class Config:
    MONGO_URI = os.environ.get(
        'MONGO_URI',
        'mongodb://localhost:27017/crewdeck'
    )
    JWT_SECRET_KEY = os.environ.get(
        'JWT_SECRET_KEY', 'crewdeck-secret-key'
    )
    UPLOAD_FOLDER = os.path.join(os.getcwd(), 'uploads')
    MAX_CONTENT_LENGTH = 20 * 1024 * 1024
    ALLOWED_EXTENSIONS = {'pdf', 'docx', 'xlsx'}
```

Лістинг файлу routes/auth.py:

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import create_access_token
from werkzeug.security import (generate_password_hash,
                               check_password_hash)
from bson import ObjectId
```

```
from datetime import datetime
from app import db

auth_bp = Blueprint('auth', __name__)

@auth_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    email = data.get('email')
    password = data.get('password')

    if not email or not password:
        return jsonify(
            {'error': 'Email та пароль є обовязковими'}
        ), 400

    if len(password) < 8:
        return jsonify(
            {'error': 'Пароль має містити мінімум 8 символів'}
        ), 400

    if db.users.find_one({'email': email}):
        return jsonify(
            {'error': 'Користувач вже існує'}
        ), 409

    hashed = generate_password_hash(password)
    db.users.insert_one({
        'email': email,
```

```
        'password': hashed,
        'role': 'user',
        'created_at': datetime.utcnow()
    })
    return jsonify({'message': 'Реєстрація успішна'}), 201

@auth_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    user = db.users.find_one({'email': data.get('email')})

    if not user or not check_password_hash(
            user['password'], data.get('password', '')):
        return jsonify({'error': 'Невірні дані'}), 401

    token = create_access_token(identity=str(user['_id']))
    return jsonify({
        'token': token,
        'role': user.get('role', 'user')
    }), 200
```

Лістинг файлу services/ocr_service.py

```
import fitz
import pytesseract
from PIL import Image
import io

def extract_text_from_pdf(file_path: str) -> str:
    doc = fitz.open(file_path)
```

```
    full_text = ""

    for page in doc:
        # Спроба вилучити машинозчитуваний текст
        text = page.get_text()
        if text.strip():
            full_text += text
        else:
            # Сторінка є скануванням — застосовуємо OCR
            pix = page.get_pixmap(dpi=300)
            img_bytes = pix.tobytes("png")
            image = Image.open(io.BytesIO(img_bytes))
            ocr_text = pytesseract.image_to_string(
                image, lang='ukr+eng'
            )
            full_text += ocr_text

    doc.close()
    return full_text
```

Лістинг файлу services/parser_service.py

```
import re
def apply_template(text: str, template: dict) -> dict:
    results = {}
    found_fields = 0
    total_fields = len(template.get('fields', []))
    for field in template.get('fields', []):
        field_name = field['name']
        keywords = field['keywords']
```

```
        value = None
        for keyword in keywords:
            escaped = re.escape(keyword)
            pattern = re.compile(
                rf'{escaped}\s*[:\-]?\s*([^\n]+)',
                re.IGNORECASE | re.MULTILINE
            )
            match = pattern.search(text)
            if match:
                raw = match.group(1).strip()
                # Пошук дати у знайденому значенні
                date_match = re.search(
                    r'\d{2}[./]\d{2}[./]\d{4}', raw
                )
                if date_match:
                    value = date_match.group(0)
                else:
                    value = raw.strip()
                if value:
                    found_fields += 1
                    break
        results[field_name] = value
    # Обчислення показника точності екстракції
    confidence = (found_fields / total_fields * 100
                  if total_fields > 0 else 0)
    return {
        'extracted_data': results,
        'confidence': round(confidence, 1)
    }
```

ДОДАТОК Б

Лістинги маршрутів API

Лістинг файлу routes/seamen.py (фрагмент):

```
from flask import Blueprint, request, jsonify
from flask_jwt_extended import (jwt_required,
                                get_jwt_identity)
from bson import ObjectId
from datetime import datetime
from app import db

seamen_bp = Blueprint('seamen', __name__)

@seamen_bp.route('', methods=['GET'])
@jwt_required()
def get_seamen():
    current_user_id = get_jwt_identity()
    query = {'owner_id': ObjectId(current_user_id)}
    status = request.args.get('status')
    if status:
        query['status'] = status
    seamen = list(db.seamen.find(query))
    for s in seamen:
        s['_id'] = str(s['_id'])
        s['owner_id'] = str(s['owner_id'])
    return jsonify(seamen), 200

@seamen_bp.route('', methods=['POST'])
@jwt_required()
```

```
def create_seaman():
    current_user_id = get_jwt_identity()
    data = request.get_json()
    if not data.get('full_name') or not data.get('rank'):
        return jsonify(
            {'error': 'full_name та rank є обовязковими'}
        ), 400
    seaman = {
        'full_name': data['full_name'],
        'rank': data['rank'],
        'birth_date': data.get('birth_date', ''),
        'document_number': data.get('document_number', ''),
        'status': data.get('status', 'available'),
        'owner_id': ObjectId(current_user_id),
        'files': [],
        'created_at': datetime.utcnow()
    }
    result = db.seamen.insert_one(seaman)
    return jsonify({
        'message': 'Профіль створено',
        'id': str(result.inserted_id)
    }), 201


@seamen_bp.route('/<id>', methods=['PUT'])
@jwt_required()
def update_seaman(id):
    current_user_id = get_jwt_identity()
    seaman = db.seamen.find_one({
        '_id': ObjectId(id),
```

```
        'owner_id': ObjectId(current_user_id)
    })
    if not seaman:
        return jsonify({'error': 'Запис не знайдено'}), 404
    data = request.get_json()
    db.seamen.update_one(
        {'_id': ObjectId(id)},
        {'$set': data}
    )
    return jsonify({'message': 'Профіль оновлено'}), 200

@seamen_bp.route('/<id>', methods=['DELETE'])
@jwt_required()
def delete_seaman(id):
    current_user_id = get_jwt_identity()
    result = db.seamen.delete_one({
        '_id': ObjectId(id),
        'owner_id': ObjectId(current_user_id)
    })
    if result.deleted_count == 0:
        return jsonify({'error': 'Запис не знайдено'}), 404
    return jsonify({'message': 'Профіль видалено'}), 200

@seamen_bp.route('/export', methods=['GET'])
@jwt_required()
def export_seamen():
    from flask import send_file
    from services.export_service import export_to_excel
    current_user_id = get_jwt_identity()
```

```
    seamen = list(db.seamen.find(
        {'owner_id': ObjectId(current_user_id)}
    ))
    output = export_to_excel(seamen)
    return send_file(
        output,
        mimetype='application/vnd.openxmlformats-'
                 'officedocument.spreadsheetml.sheet',
        as_attachment=True,
        download_name='seamen_export.xlsx'
    )
```

Лістинг файлу routes/files.py (фрагмент):

```
from flask import Blueprint, request, jsonify
from flask import current_app, send_file
from flask_jwt_extended import (jwt_required,
                                get_jwt_identity)
from werkzeug.utils import secure_filename
from bson import ObjectId
from datetime import datetime
from app import db
from services.ocr_service import extract_text_from_pdf
from services.parser_service import apply_template
import os
import time

files_bp = Blueprint('files', __name__)
ALLOWED_TYPES = {
    'application/pdf',
```

```
    'application/vnd.openxmlformats-officedocument'
    '.wordprocessingml.document',
    'application/vnd.ms-excel'
}
@files_bp.route('/upload', methods=['POST'])
@jwt_required()
def upload_file():
    current_user_id = get_jwt_identity()
    if 'file' not in request.files:
        return jsonify({'error': 'Файл не знайдено'}), 400
    file = request.files['file']
    if file.content_type not in ALLOWED_TYPES:
        return jsonify(
            {'error': 'Недозволений тип файлу'}
        ), 415
    filename = secure_filename(file.filename)
    unique_name = f"{int(time.time())}_{filename}"
    filepath = os.path.join(
        current_app.config['UPLOAD_FOLDER'], unique_name
    )
    file.save(filepath)
    file_doc = {
        'filename': filename,
        'filepath': filepath,
        'status': 'processing',
        'owner_id': ObjectId(current_user_id),
        'uploaded_at': datetime.utcnow()
    }
    seaman_id = request.form.get('seaman_id')
```

```
    if seaman_id:
        file_doc['seaman_id'] = ObjectId(seaman_id)
    file_id = db.files.insert_one(file_doc).inserted_id
    try:
        text = extract_text_from_pdf(filepath)
        template_id = request.form.get('template_id')
        extracted = {}
        confidence = 0
        if template_id:
            template = db.templates.find_one(
                {'_id': ObjectId(template_id)}
            )
            if template:
                result = apply_template(text, template)
                extracted = result['extracted_data']
                confidence = result['confidence']
                db.extraction_results.insert_one({
                    'file_id': file_id,
                    'template_id': ObjectId(template_id),
                    'extracted_data': extracted,
                    'confidence': confidence,
                    'created_at': datetime.utcnow()
                })
        if seaman_id:
            db.seamen.update_one(
                {'_id': ObjectId(seaman_id)},
                {'$push': {'files': {
                    'file_id': str(file_id),
                    'filename': filename,
```

```
                'uploaded_at': datetime.utcnow()
            }}}
        )
        db.files.update_one(
            {'_id': file_id},
            {'$set': {'status': 'completed'}}
        )
        return jsonify({
            'message': 'Файл оброблено',
            'file_id': str(file_id),
            'extracted_data': extracted,
            'confidence': confidence
        }), 202
    except Exception as e:
        db.files.update_one(
            {'_id': file_id},
            {'$set': {'status': 'failed', 'error': str(e)}}
        )
        return jsonify({'error': str(e)}), 500
@files_bp.route('/download/<file_id>', methods=['GET'])
@jwt_required()
def download_file(file_id):
    current_user_id = get_jwt_identity()
    file_doc = db.files.find_one({
        '_id': ObjectId(file_id),
        'owner_id': ObjectId(current_user_id)
    })
    if not file_doc:
        return jsonify({'error': 'Файл не знайдено'}), 404
```

```
    filepath = file_doc['filepath']
    if not os.path.exists(filepath):
        return jsonify(
            {'error': 'Файл не знайдено на диску'}
        ), 404
    return send_file(
        filepath,
        as_attachment=True,
        download_name=file_doc['filename']
    )
@files_bp.route('/<file_id>', methods=['DELETE'])
@jwt_required()
def delete_file(file_id):
    current_user_id = get_jwt_identity()
    file_doc = db.files.find_one({
        '_id': ObjectId(file_id),
        'owner_id': ObjectId(current_user_id)
    })
    if not file_doc:
        return jsonify({'error': 'Файл не знайдено'}), 404
    if os.path.exists(file_doc['filepath']):
        os.remove(file_doc['filepath'])
    db.files.delete_one({'_id': ObjectId(file_id)})
    if file_doc.get('seaman_id'):
        db.seamen.update_one(
            {'_id': file_doc['seaman_id']},
            {'$pull': {'files': {'file_id': file_id}}}
        )
    return jsonify({'message': 'Файл видалено'}), 200
```

ДОДАТОК В

Лістинги клієнтської частини

Лістинг файлу services/api.js:

```
import axios from 'axios';

const api = axios.create({
  baseURL: 'http://localhost:5000/api',
});
api.interceptors.request.use(config => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});
api.interceptors.response.use(
  response => response,
  error => {
    if (error.response?.status === 401) {
      localStorage.removeItem('token');
      localStorage.removeItem('userEmail');
      localStorage.removeItem('userRole');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
export default api;
```

Лістинг файлу context/AuthContext.js:

```
import React, { createContext, useState,
                useContext } from 'react';

const AuthContext = createContext(null);

export const AuthProvider = ({ children }) => {
  const [token, setToken] = useState(
    localStorage.getItem('token')
  );

  const login = (newToken) => {
    setToken(newToken);
    localStorage.setItem('token', newToken);
  };

  const logout = () => {
    setToken(null);
    localStorage.removeItem('token');
    localStorage.removeItem('userEmail');
    localStorage.removeItem('userRole');
  };
  return (
    <AuthContext.Provider value={{ token, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
export const useAuth = () => useContext(AuthContext);
```

Лістинг файлу pages/SeamenManagement.js (фрагмент):

```
const uploadFiles = async (files) => {
  setIsUploading(true);
  setMessage('');
  setExtractedData(null);
  for (const file of files) {
    const formData = new FormData();
    formData.append('file', file);
    if (selectedSeaman)
      formData.append('seaman_id', selectedSeaman._id);
    if (selectedTemplate)
      formData.append('template_id', selectedTemplate);
    try {
      const res = await api.post('/files/upload', formData);
      setMessage(
        `ok:Файл "${file.name}" успішно оброблено`
      );
      if (res.data.extracted_data &&
          Object.keys(res.data.extracted_data).length > 0) {
        setExtractedData({
          data: res.data.extracted_data,
          confidence: res.data.confidence
        });
      }
    } catch {
      setMessage(
        `err:Помилка при обробці файлу "${file.name}"`
      );
    }
```

```
  }
  setIsUploading(false);
  fetchSeamen();
};

const handleCreate = async () => {
  if (!newSeaman.full_name || !newSeaman.rank) return;
  try {
    await api.post('/seamen', newSeaman);
    setShowModal(false);
    setNewSeaman({
      full_name: '', rank: '',
      birth_date: '', document_number: ''
    });
    fetchSeamen();
  } catch { }
};

const handleStatusChange = async (id, newStatus) => {
  try {
    await api.put(`/seamen/${id}`, { status: newStatus });
    fetchSeamen();
  } catch { }
};

const handleUpdateSeaman = async () => {
  if (!editingSeaman) return;
  try {
    await api.put(
```

```
      `/seamen/${editingSeaman._id}`, editingData
    );
    setEditingSeaman(null);
    fetchSeamen();
  } catch { }
};

const handleDeleteFile = async (seamanId, fileId) => {
  if (!window.confirm('Видалити цей файл?')) return;
  try {
    await api.delete(`/files/${fileId}`);
    fetchSeamen();
  } catch { }
};
```

Лістинг файлу pages/ExtractionTemplates.js (фрагмент):

```
const handleSubmit = async () => {
  const payload = {
    name: formData.name,
    category: formData.category,
    fields: formData.fields.map(f => ({
      name: f.name,
      keywords: f.keywords.split(',')
        .map(k => k.trim())
        .filter(Boolean)
    }))
  };
  try {
    if (editingTemplate) {
```

```
      await api.delete(
        `/templates/${editingTemplate._id}`
      );
    }
    await api.post('/templates', payload);
    setShowForm(false);
    setEditingTemplate(null);
    setFormData({
      name: '', category: 'general',
      fields: [{ name: '', keywords: '' }]
    });
    fetchTemplates();
  } catch { }
};

const openEdit = (t) => {
  setEditingTemplate(t);
  setFormData({
    name: t.name,
    category: t.category,
    fields: t.fields.map(f => ({
      name: f.name,
      keywords: Array.isArray(f.keywords)
        ? f.keywords.join(', ')
        : f.keywords
    }))
  });
  setShowForm(true);
};
```

```
 const handleDelete = async (id) => {
   if (!window.confirm('Видалити шаблон?')) return;
   try {
     await api.delete(`/templates/${id}`);
     fetchTemplates();
   } catch { }
 };
```