**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ
кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

“До захисту допущено”
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА

“___”_____2024 р.

# Дипломна робота

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою**
***“Цифрові технології в енергетиці”***
**зі спеціальності *122 “Комп’ютерні науки”***

**на тему: Формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображенні**

Виконав:
студент IV курсу, групи ТР-03
БАГІНСЬКИЙ Михайло Олександрович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник: *професор каф. цифрових технологій в енергетиці*
*професор, д.т.н., Володимир СЛІПЧЕНКО* _____
(посада, вчене звання ,науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Рецензент:
_____
_____ _____
(посада, вчене звання, науковий ступінь, ім’я, ПРІЗВИЩЕ) (підпис)

Н.контроль: *асистент Микита ГОЛОВАКІН* _____
(посада, ім’я, ПРІЗВИЩЕ) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ АТОМНОЇ ТА ТЕПЛОВОЇ ЕНЕРГЕТИКИ

Кафедра ЦИФРОВИХ ТЕХНОЛОГІЙ В ЕНЕРГЕТИЦІ

Рівень вищої освіти – перший (бакалаврський)
спеціальність 122 “Комп’ютерні науки”
Освітньо-професійна програма “Цифрові технології в енергетиці”

ЗАТВЕРДЖУЮ
Завідувач кафедри ЦТЕ
_____ Наталія АУШЕВА_

«__»_____2024 р.

**ЗАВДАННЯ**
**на дипломну роботу студенту**

**БАГІНСЬКОМУ Михайлу Олександровичу**
(прізвище, ім’я, по батькові)

1. Тема роботи **Формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображенні**

керівник роботи ***Сліпченко Володимир Георгійович, д.т.н., професор***
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «___»_____ 202__ р. №_____

2. Термін подання роботи студентом **07.06.2024 року**

3. Вихідні дані до роботи мова програмування JavaScript, фреймворк Express, СКБД PostgreSQL, середовище розробки VS Code

4. Перелік питань, які потрібно розробити
1) провести аналіз серед наявних аналогів систем для розпізнання тексту на зображені
2) аргументувати вибрані інструменти, технології та алгоритми для реалізації програмного забезпечення
3) побудувати архітектуру програмного забезпечення, бази даних
4) створити прикладний програмний інтерфейс
5) представити процес застосування інтерфейсу

5. Орієнтований перелік ілюстративного матеріалу діаграми, що демонструють роботу алгоритмів інтерфейсу, архітектуру системи, бази даних, взаємодію користувачів із системою, класи програмного забезпечення; приклади роботи з програмним продуктом.

6. Дата видачі завдання «15» вересня 2023 р.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів виконання дипломної роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| 1. | Затвердження теми роботи | | |
| 2. | Вивчення та аналіз задачі | 17-20.04.24 | |
| 3. | Розробка архітектури та загальної структури системи | 21-25.04.24 | |
| 4. | Розробка частин окремих підсистем | 25.04-02.05.24 | |
| 5. | Програмна реалізація системи | 03-07.05.24 | |
| 6. | Оформлення пояснювальної записки | 08-12.05.24 | |
| 7. | Захист програмного продукту | 13-17.05.24 | |
| 8. | Передзахист | 03-06.06.24 | |
| 9. | Подання готової роботи на кафедру | 07.06.2024 | |
| 10. | Захист | 17-21.06.2024 | |

Студент _____ **Михайло БАГІНСЬКИЙ**
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ **Володимир СЛІПЧЕНКО**
(підпис) (ім'я, ПРІЗВИЩЕ)

# АНОТАЦІЯ

Дипломна робота виконана на 52 сторінках, містить 38 ілюстрацій, 1 таблицю, 1 додаток та 25 джерел в переліку посилань.

Мета роботи – створення програмного забезпечення для формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображенні.

Методи та засоби: мова програмування JavaScript, фреймворк Express, СУБД PostgreSQL, бібліотеки для розпізнання тексту Tesseract.js та Google Cloud Vision.

Результат – система формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображенні.

Ключові слова: РОЗПІЗНАННЯ ТЕКСТУ, OCR, БАЗА ДАНИХ, ЛАБОРАТОРНІ АНАЛІЗИ.

# ABSTRACT

The thesis is 52 pages long, contains 38 illustrations, 1 table, 1 appendix, and 25 references.

The purpose of the work is to create software for the formation of a database of laboratory analyzes by recognizing text in an image.

Methods and tools: JavaScript programming language, Express framework, PostgreSQL database, Tesseract.js and Google Cloud Vision libraries for text recognition.

The result is a system for creating a database of laboratory analyzes by recognizing text in an image.

Keywords: TEXT RECOGNITION, OCR, DATABASE, LABORATORY TESTS.

# ЗМІСТ

# ВСТУП

У світі медичних досліджень, де обсяг інформації зростає з кожним днем, важливість своєчасного та ефективного збору, збереження та аналізу результатів лабораторних аналізів набуває надзвичайної ваги. З використанням технологій розпізнавання тексту на зображеннях відкриваються нові можливості для автоматизації цих процесів. Це дозволяє значно підвищити швидкість обробки інформації, а використання бази даних для збереження інформації забезпечує ефективне управління медичними даними.

Тема роботи є актуальною з декількох причин. По-перше, це економія часу та зусиль, адже ручний перенос даних з паперових аналізів до електронних баз даних є трудомістким та витратним процесом. Розпізнавання тексту на зображеннях дозволяє автоматизувати цей процес, заощаджуючи значні ресурси.

По-друге, це підвищення точності даних, оскільки введення даних вручну схильне до помилок, особливо при тривалій роботі з великими обсягами аналізів. Розпізнавання тексту на зображеннях мінімізує кількість людських помилок, забезпечуючи вищу точність даних.

Ще однією з причин є довгострокове зберігання даних та їх доступність. Цифрові бази даних забезпечують довготривале та надійне зберігання лабораторних аналізів, а також полегшують доступ до них для подальшої роботи.

Тому, формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображеннях є актуальним завданням для автоматизації та оптимізації роботи медичних закладів.

Метою даної роботи є розробка системи для розпізнавання тексту на зображеннях лабораторних аналізів та формування бази даних з отриманих результатів.

Щоб досягти поставленої мети потрібно вирішити наступні завдання:

1. Аналіз існуючих методів та алгоритмів розпізнавання тексту на зображеннях.

2. Обрати та налаштувати відповідні моделі та бібліотеки для розпізнавання тексту.

3. Розробка алгоритму для структурування даних з розпізнаного тексту.

4. Створення бази даних для зберігання результатів лабораторних аналізів.

5. Інтегрування всіх компонентів в єдину систему.

6. Тестування ефективності та точності системи на реальних зображеннях.

# 1 АНАЛІЗ ЗАДАЧІ ПОБУДОВИ СИСТЕМИ ДЛЯ ФОРМУВАННЯ БАЗИ ДАНИХ ЛАБОРАТОРНИХ АНАЛІЗІВ ШЛЯХОМ РОПІЗНАННЯ ТЕКСТУ НА ЗОБРАЖЕНІ

У цьому розділі розглянуто задачу побудови системи для формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображеннях. Розглянуті ключові аспекти, які необхідно врахувати при розробці такої системи. Сформовано основні задачі для вирішення та виділено основний функціонал, що потрібно реалізувати. Розглянуто методи розпізнання тексту на зображеннях. Розглянуто вибір мови програмування та бази даних.

## 1.1 Постановка задачі

У цій роботі розглядається проблематика збору та аналізу результатів лабораторних аналізів, за допомогою методів розпізнання тексту на зображені. Важливість цієї проблематики полягає у необхідності оперативного доступу до даних для медичних досліджень та діагностики. Застосування штучного інтелекту у цьому контексті може суттєво полегшити процес формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображеннях.

Основною метою розробки такої системи є створення ефективного інструменту, який дозволить автоматизувати процес збору та класифікації інформації з зображень у структуровані дані, що будуть придатними для подальшого зберігання та аналізу в базі даних. Заповнення бази даних за допомогою системи автоматичного розпізнавання тексту на зображеннях лабораторних аналізів стає важливим для забезпечення доступності та управління медичною інформацією. Система повинна забезпечувати досить високу точність розпізнавання тексту, можливість редагування інформації до та після завантаження її до бази даних, а також захистити доступ до конфіденційної інформації пацієнтів.

Основними викликами при реалізації такої системи є підвищення точності розпізнавання, обробка різних форматів документів та забезпечення безпеки даних.

Підвищення точності розпізнавання можна досягти за рахунок покращення якості зображень та використання передових моделей машинного навчання.

Обробка різних форматів документів потребує налаштування системи для роботи з документами різних форматів та мов.

Забезпечення безпеки даних є критичним завданням, яке вирішується шляхом використання шифрування та інших заходів захисту.

Задачі для вирішення:

1. Проєктування процесу збору, збереження та аналізу даних.
2. Пошук методів розпізнання тексту, бази даних для збереження даних.
3. Розробка програмного забезпечення для збору, збереження та аналізу даних.
4. Оцінка ефективності рішення.

Розроблений програмний продукт має надавати наступні основні функції:

1. Завантаження зображень лабораторних аналізів.
2. Розпізнання тексту на зображеннях.
3. Завантаження аналізу до бази даних.
4. Можливість редагування завантажених аналізів в базі даних.

## 1.2 Методи розпізнавання тексту

Для розпізнавання тексту на зображенні використовують оптичне розпізнавання символів (OCR). OCR представляє собою суміш технологій та методів, які використовуються для ідентифікації та вилучення тексту з неструктурованих документів, наприклад, зображення, знімки екрану, та фізичні паперові документи, з високою точністю використовуючи штучний інтелект та комп’ютерний зір [1]. Приклад використання OCR зображено на рисунку 1.1.

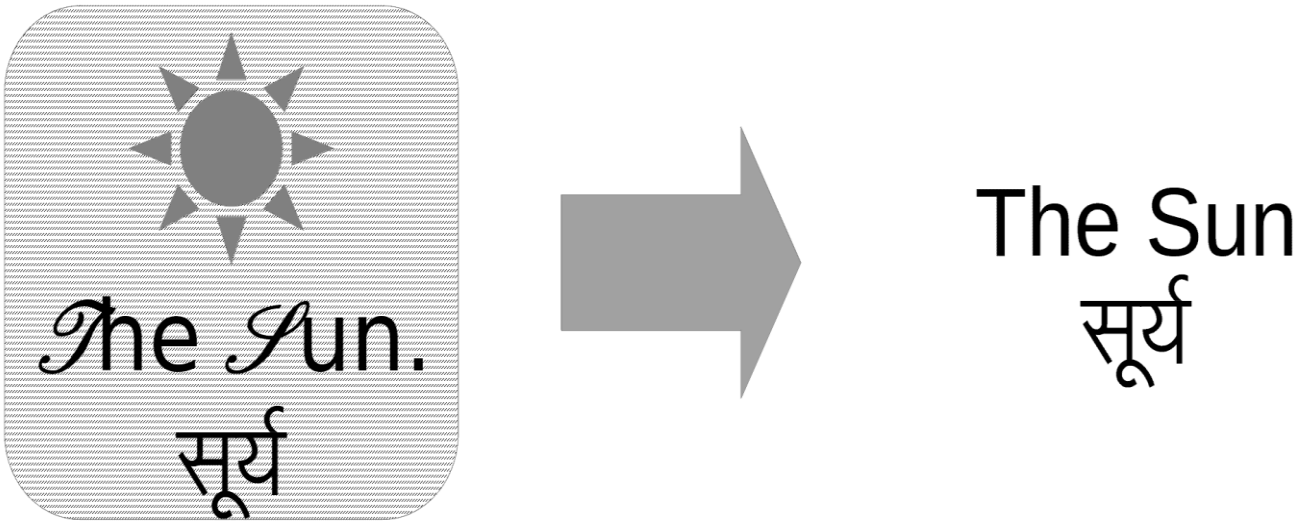


Рисунок 1.1 – Приклад використання OCR

Технологія OCR широко застосовуються в різних сферах, таких як цифровізація документів, автоматизація бізнес-процесів, обробка рукописних текстів, та багато інших, де потрібно ефективне і точне перетворення текстової інформації зі сканованих або фотографованих документів у редагований формат.

Існує декілька основних методів розпізнавання тексту на зображеннях [2]:

Метод шаблонів (Pattern Recognition). Цей метод базується на співставленні символів з наперед визначеними шаблонами. Кожен символ порівнюється з базою шаблонів і класифікується як найбільш подібний шаблон. Цей метод добре працює для друкованого тексту з фіксованим шрифтом, але не дуже ефективний для рукописного тексту.

Метод ознак (Feature Extraction). В цьому методі символи аналізуються на основі виділених ознак, таких як лінії, кути, петлі тощо. Ці ознаки порівнюються з ознаками, що зберігаються в базі даних для класифікації символів. Він краще працює для рукописного тексту, але вимагає ретельної підготовки даних.

Штучні нейронні мережі. Цей метод використовує глибоке навчання для автоматичного виявлення патернів на зображеннях тексту. Нейронні мережі навчаються на великих наборах даних зображень тексту для розпізнавання символів. Вони показують високу точність для різноманітних шрифтів та стилів, але вимагають великих обсягів даних для навчання.

Гібридні методи. Гібридні методи поєднують різні техніки, такі як використання попередньо визначених правил для сегментації тексту, а потім застосування нейронних мереж для розпізнавання окремих символів. Такі підходи часто демонструють високу точність.

Сучасні системи OCR також можуть використовувати додаткові методи обробки, такі як корекція похилості, видалення шумів, згладжування тощо для підвищення якості розпізнавання. Вибір методу залежить від типу тексту (друкований, рукописний), мови та необхідної точності розпізнавання.

## 1.3 Огляд засобів розробки

### 1.3.1 Вибір мови програмування

Вибір мови програмування є важливим у розробці програмного забезпечення, оскільки від цього залежить ефективність розробки, продуктивність додатка та легкість підтримки коду [3]. Приклад мов програмування, які підходять для цього проєкту:

Python є однією з найпопулярніших мов програмування завдяки своїй простоті, читабельності та широкому спектру бібліотек і фреймворків для різних завдань. Переваги:

1. Велика кількість бібліотек для обробки зображень та OCR, таких як OpenCV та Tesseract.
2. Підтримка бібліотек для роботи з базами даних, таких як SQLAlchemy та psycopg2.
3. Простота синтаксису та швидкість розробки.

Недоліки: може бути менш продуктивним порівняно з мовами компіляції (наприклад, C++).

Java є однією з найпопулярніших мов програмування, яка використовується для розробки масштабованих і надійних додатків. Переваги:

1. Висока продуктивність та надійність.

2. Велика кількість фреймворків для роботи з веб-додатками та базами даних (Spring, Hibernate).

3. Кросплатформенність.

Недоліки: більш складний синтаксис та довший цикл розробки порівняно з Python.

C# є потужною та широко використовуваною мовою програмування, яка працює в середовищі .NET. Вона забезпечує високий рівень продуктивності, надійність та широкий спектр бібліотек для різних завдань. Переваги:

1. Тісна інтеграція з Windows середовищем та службами Microsoft.

2. Потужний фреймворк .NET для розробки різних типів додатків.

3. Висока продуктивність.

Недоліки: залежність від екосистеми Microsoft, що може бути обмеженням на інших платформах.

C++ є мовою програмування загального призначення з високою продуктивністю та низькорівневим доступом до системних ресурсів. Переваги:

1. C++ є високопродуктивною мовою програмування, що дозволяє створювати швидкі та ефективні програми.

2. Можливість прямого управління пам'яттю та ресурсами дозволяє оптимізувати додаток для високої продуктивності.

3. C++ код може бути компільований на різних платформах без змін, що дозволяє створювати кросплатформенні додатки.

Недоліки: C++ є складною мовою програмування, яка вимагає глибокого розуміння пам'яті та низькорівневих концепцій. Відсутність автоматичного управління пам'яттю може призводити до потенційних проблем з витоком пам'яті та викликати помилки.

JavaScript є популярною мовою програмування, яка використовується не тільки для розробки клієнтської частини веб-додатків, але й для серверної частини завдяки платформі Node.js. Переваги:

1. Використання однієї мови для обох частин додатка спрощує розробку та підтримку коду.

2. Node.js побудований на основі V8 JavaScript Engine від Google, що забезпечує високу продуктивність та швидкість виконання.

3. Асинхронна архітектура Node.js дозволяє ефективно обробляти багато одночасних запитів.

4. Велика кількість пакетів у npm (Node Package Manager) дозволяє легко додавати необхідний функціонал, такий як обробка зображень, робота з базами даних та інше.

Недоліки: Node.js, як і будь-яка інша асинхронна платформа, може виявити себе неефективним у випадках, коли потрібна велика обчислювальна потужність або інтенсивна операція вводу-виводу. JavaScript, будучи мовою інтерпретованою, зазвичай має меншу швидкість виконання порівняно з компільованими мовами, такими як C++ або Java.

### 1.3.2 Вибір бази даних

База даних повинна забезпечувати надійне зберігання та швидкий доступ до даних лабораторних аналізів. Важливими критеріями вибору є продуктивність, масштабованість та підтримка складних запитів [4].

MySQL. Переваги:

1. Висока продуктивність та надійність.
2. Підтримка різних типів даних та складних запитів.
3. Велика спільнота та кількість ресурсів.

Недоліки: Обмежена підтримка деяких розширених функцій порівняно з PostgreSQL. Переваги:

1. Підтримка складних запитів та великої кількості типів даних.
2. Висока надійність та масштабованість.
3. Можливість розширення функціональності за допомогою сторонніх модулів.

Недоліки: складніше налаштування та адміністрування порівняно з MySQL.

MongoDB. Переваги:

1. Гнучкість у роботі з неструктурованими даними.
2. Висока продуктивність для операцій з великими обсягами даних.

3. Підтримка горизонтального масштабування.

Недоліки: відсутність підтримки складних транзакцій та обмежена підтримка складних запитів.

# 2 ТЕОРЕТИЧНІ ВІДОМОСТІ ПРО МЕТОДИ РОЗПІЗНАННЯ ТЕКСТУ

Розпізнавання тексту є одним з ключових компонентів розробленої системи, що дозволяє автоматизувати обробку та аналіз великих обсягів текстової інформації. Цей розділ описує алгоритми, які використовуються при цьому процесі. Також в розділі описуються існуючі системи розпізнавання тексту.

## 2.1 Початок використання розпізнання тексту

Витоки технології розпізнавання тексту можна простежити до появи телеграфу. Насправді під час Першої світової війни вчений Емануель Голдберг досяг значних успіхів у цій галузі, винайшовши систему, здатну перетворювати символи на телеграфний код. Це видатне досягнення стало можливим завдяки попереднім дослідженням Голдберга, які включали новаторське створення електронної системи пошуку документів у 1920-х роках, що стало першою у своєму роді.

У 1974 році Рей Курцвейл заснував компанію Kurzweil Computer Products, Inc., де він очолив розробку універсальної системи оптичного розпізнавання символів (OCR). Ця новаторська технологія мала надзвичайну здатність ідентифікувати текст, написаний практично будь-яким шрифтом. Визнаючи його потенціал для допомоги людям з вадами зору, Курцвейл винайшов пристрій для читання, який перетворював текст на звукову мову.

Оцифрування історичних газет на початку 1990-х років призвело до появи технології оптичного розпізнавання тексту, яка швидко набула популярності. З того часу в цій галузі було досягнуто багато успіхів [5, 6].

## 2.2 Аналіз процесу розпізнання тексту

Процес розпізнавання тексту можна поділити на кілька ключових етапів, що зображені на рисунку 2.1.

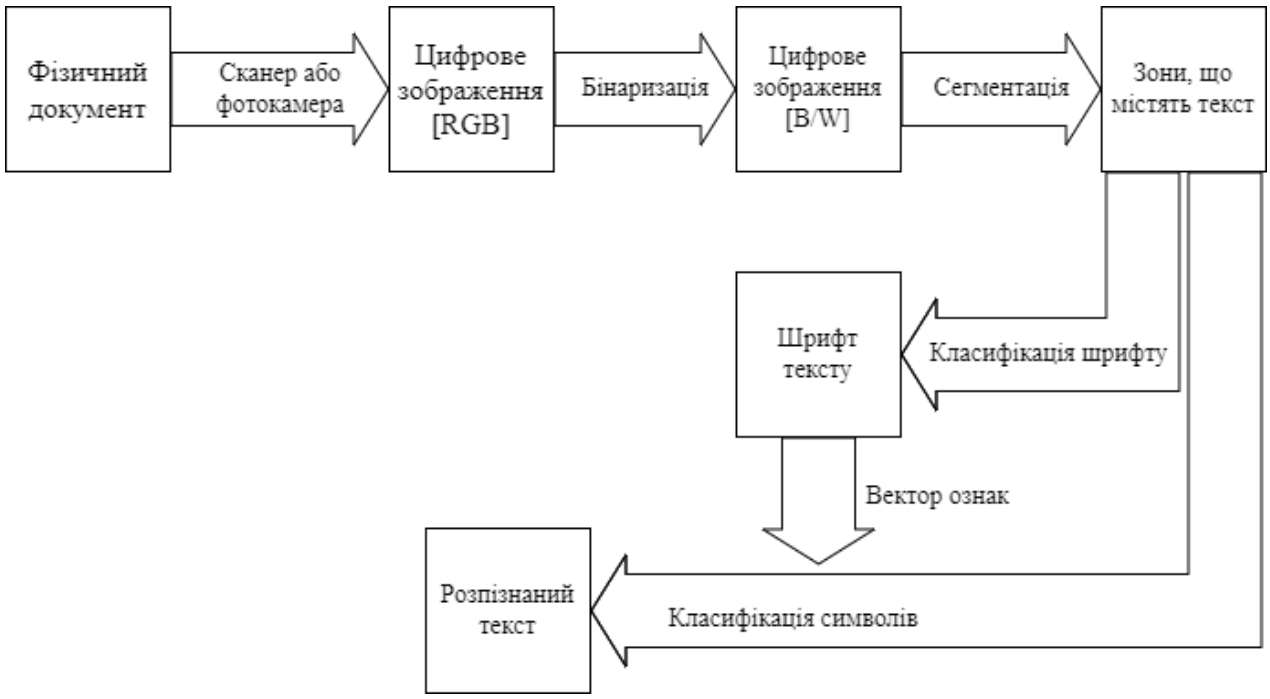


Рисунок 2.1 – Етапи роботи OCR

Для перетворення фізичного документа в цифровий формат використовується камера або сканер. Отриманий документ потім піддається програмному забезпеченню OCR, яке перетворює його на двоколірну або чорно-білу версію. Система переходить до перевірки сканованого або растрового зображення, розрізняючи світлі та темні ділянки. Світлі ділянки розпізнаються як фон, а темні – як символи.

Наступна дія передбачає ідентифікацію буквених або цифрових символів за допомогою аналізу затінених областей. Під час цієї фази система спрямовує свою увагу на окремі символи, слова або частини тексту. Для розпізнавання символів зазвичай використовуються або алгоритми розпізнавання шаблонів, або розпізнавання ознак.

Розпізнавання шаблонів використовується для порівняння та ідентифікації символів у відсканованому документі або файлі зображення. Для використання цього алгоритму потрібно надати програмі розпізнавання тексту приклади тексту різних шрифтів і форматів.

Другий алгоритм називається – розпізнаванням ознак. В цьому алгоритмі OCR використовує правила, що стосуються характеристик певної літери або цифри, для розпізнавання символів у відсканованому документі. Характеристики, що застосовуються, включають в себе кількість вигнутих, перехресних або кутових ліній.

Структура зображення або документа так само вивчається програмою розпізнавання тексту. Вона розділяє сторінку на частини такі, які текстові блоки, таблиці та графіку. Спочатку система розділяє текстові блоки, утворюючи слова, а потім і на окремі символи. Після ідентифікації символів алгоритм порівнює їх з колекцією шаблонних зображень. Після того, як програма перебирає всі можливі збіги, вона виводить розпізнаний текст [7].

### 2.2.1 Сканування зображення

Сканування зображення - це процес перетворення фізичного документа в цифровий формат за допомогою спеціального обладнання, такого як камера або сканер.

Сканери бувають різних типів: планшетні, протяжні та ручні. Планшетні сканери є найбільш поширеними і забезпечують високу якість зображень, оскільки документ кладеться на скляну поверхню, і скануюча голівка проходить під ним, зчитуючи інформацію. Протяжні сканери використовуються для сканування окремих аркушів або документів великого обсягу, працюють швидше, але можуть бути менш точними для товстих або пошкоджених документів. Ручні сканери - це портативні пристрої, які користувач переміщує над поверхнею документа вручну, вони зручні для сканування великих об’єктів або в умовах обмеженого простору. Камери, особливо сучасні цифрові камери та смартфони з високою роздільною здатністю, також можуть використовуватися для створення цифрових копій документів. Для досягнення найкращих результатів при використанні камер

важливо дотримуватись певних правил, таких як правильне освітлення та стабільність камери.

### 2.2.2 Бінаризація зображення

Ключова ідея полягає в тому, щоб видалити нетекстові артефакти з графічного документа.

Часто документи зберігаються в пам'яті комп'ютера у вигляді рівня сірого, який має максимум 256 різних значень сірого в діапазоні від 0 до 255. Кожне значення сірого в палітрі відтінків сірого дає свій колір. Якщо зображення в документі повинно містити якусь інформацію, потрібно повторити процес багато разів для різних кольорів. Двійкове зображення є більш корисним, оскільки воно може скоротити час, необхідний для вилучення фрагмента зображення.

Існує можливість перетворити будь-яке багатотонне зображення у відтінках сірого на чорно-біле за допомогою техніки бінаризації (двоколірне зображення). Приклад використання зображено на рисунку 2.2.


Рисунок 2.2 – Приклад використання бінаризації

Щоб розпочати процес бінаризації, потрібно визначити порогове значення відтінків сірого і те, чи має піксель певне значення сірого. Якщо значення сірого

пікселя перевищує порогове значення, піксель стає білим. Аналогічно, пікселі стають чорними, якщо їхнє значення сірого менше порогового [8].

### 2.2.3 Класифікація шрифту

Мета цього процесу – ідентифікація шрифту, яким написаний текст. Ідентифікація шрифту включає в себе визначення того, чи написаний текст від руки, а в деяких випадках, чи написаний він конкретною особою.

Ідентифікація мови тексту є частиною його стильової класифікації. Це важливо, оскільки мови, що належать до різних регіонів світу (наприклад, Азії), мають особливі символи або стилі написання, притаманні лише цій мові.

Напрямок письма або спосіб з'єднання символів у слова може відрізнятися в різних мовах. Ми пишемо арабською справа наліво, а в деяких індійських мовах з'єднуються в рядок кілька символів, щоб утворити слово. Приклад написання тексту зображено на рисунку 2.3.


Рисунок 2.3 – Приклад написання тексту

Ідентифікація шрифту написання має важливе значення для вибору алгоритму розпізнавання символів, який найкраще підходить для фрагмента тексту.

Одним із способів ідентифікації стилю є виділення декількох випадкових фрагментів тексту в якості тестових зразків. Далі перетворюємо ці тестові зразки на вектори ознак, які однозначно ідентифікують стильові особливості тексту.

Тепер можна порівняти ці вектори ознак з наявною базою даних векторів ознак, які ідентифікують певну мову, шрифт або почерк. Таким чином, можна ідентифікувати стиль повного тексту як найбільш ймовірний збіг з вектором ознак тестових зразків [8].

### 2.2.4 Класифікація символів

На цьому останньому етапі роботи системи розпізнавання відбувається ідентифікація окремих символів. На основі стилю, визначеного на попередньому етапі, виділяються окремі символи з графічного документа (класифікація стилів).

Далі символи розділяються на сегменти і використовуються їхні геометричні властивості для ідентифікації (виявлення ознак) або використовується модель розпізнавання, навчена на попередній базі даних (розпізнавання образів). Приклад зображено на рисунку 2.4.

Можна ще більше вдосконалити результати розпізнавання за допомогою словника, щоб перетворити безглузді слова на їхні найближчі правильні варіанти. Це схоже на функцію автоматичного виправлення орфографії на деяких пристроях сьогодні.

Фізичне розміщення тексту на зображенні також надає інформацію про те, яку підгрупу символів слід очікувати. Прикладом цього є те, що в розділі телефонних номерів очікується послідовність цифр, тоді як ім'я - це здебільшого рядок нецифрових символів.

Контекстна інформація слугує ще одним уточненням, оскільки речення в певній мові будується за певними правилами. Це схоже на граматичну перевірку, яку виконують сучасні текстові редактори.

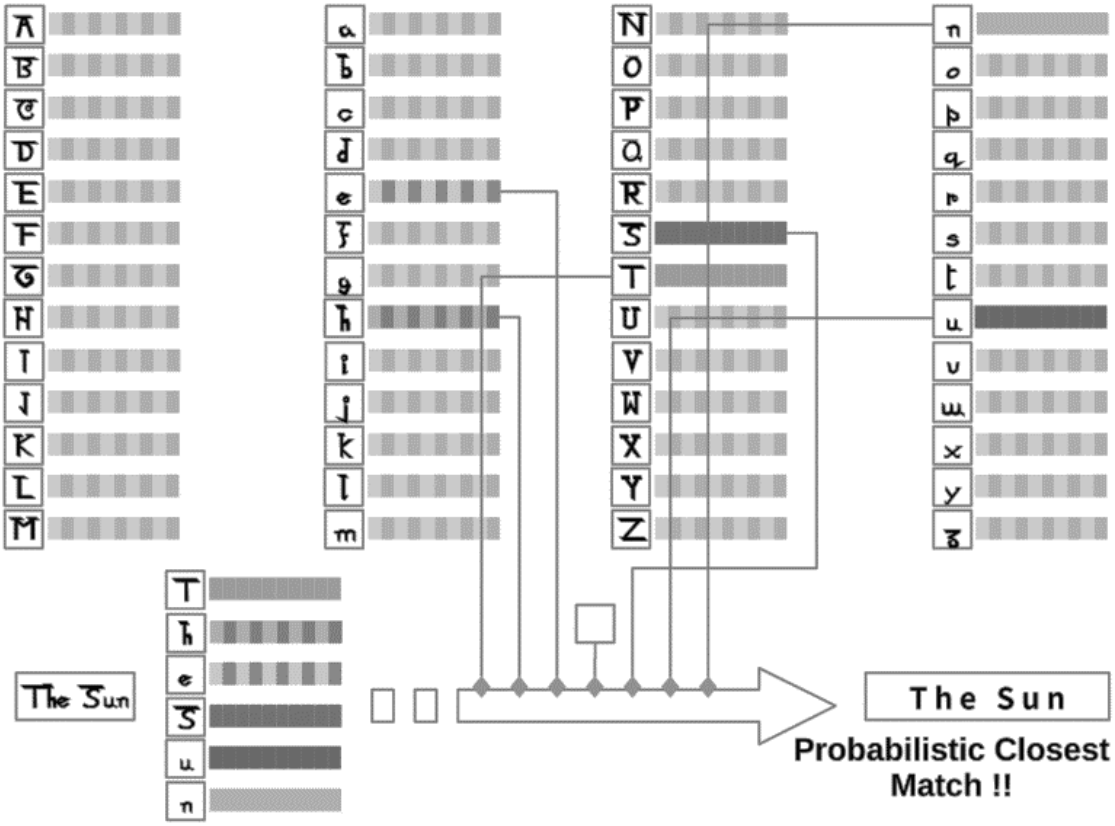


Рисунок 2.4 – Приклад ідентифікації символів

Глибоке навчання та інші вдосконалені моделі розпізнавання символів працюють над кількома іншими факторами навколишнього середовища, уточнюючи ідентифікований текст до кінцевого результату роботи системи розпізнавання символів [9].

### 2.3 Аналіз існуючих систем

Прямих аналогів системи формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображеннях наразі не існує. Існуючі системи, такі як ABBYY FineReader, Tesseract OCR, Google Cloud Vision, Amazon Textract та Adobe Acrobat, надають функціонал для завантаження зображень та отримання звичайного неструктурованого тексту. Вони не забезпечують автоматичного

структурованого збереження даних у базі даних з урахуванням специфіки лабораторних аналізів.

ABBYY FineReader є одним з найвідоміших комерційних рішень для розпізнавання тексту. Цей програмний продукт підтримує широкий спектр мов і типів документів. FineReader забезпечує високу точність розпізнавання тексту завдяки використанню передових алгоритмів OCR та машинного навчання. Однак, висока вартість ліцензії може бути недоліком для деяких користувачів [10].

Tesseract OCR - це безкоштовний і відкритий інструмент для розпізнавання тексту, розроблений компанією Google. Він підтримує багато мов і може бути інтегрований в різні системи завдяки своїй гнучкості та доступності. Tesseract добре підходить для проєктів з обмеженим бюджетом, проте його точність може поступатися комерційним рішенням у складних випадках [11].

Google Cloud Vision є хмарним сервісом, який надає можливості для розпізнавання тексту на зображеннях, а також інші функції обробки зображень, такі як розпізнавання об’єктів і облич. Використання хмарних технологій дозволяє легко масштабувати систему, проте залежність від інтернет-з’єднання і можливі питання конфіденційності даних можуть бути недоліками [12].

Amazon Textract - це ще одне хмарне рішення, яке надає можливості для автоматизованого розпізнавання тексту та аналізу документів. Textract може витягувати текст, форми і таблиці з різних типів документів, що робить його універсальним інструментом для медичних установ. Вартість використання та конфіденційність даних також можуть бути важливими факторами для користувачів [13].

Adobe Acrobat пропонує функції OCR для перетворення сканованих документів у редагований текст. Цей інструмент добре інтегрується з іншими продуктами Adobe і може бути корисним для систем, які вже використовують екосистему Adobe. Однак, подібно до ABBYY FineReader, вартість ліцензії може бути високою [14].

Розроблена система відрізняється від наявних рішень тим, що вона включає повний цикл обробки лабораторних аналізів: від попередньої обробки зображень і розпізнавання тексту до верифікації та автоматичного структурованого збереження

даних у базі даних. Це дозволяє забезпечити високу точність і ефективність обробки, а також спрощує доступ і аналіз даних для медичних працівників. Існуючі системи, хоча і пропонують потужні інструменти для OCR, не надають спеціалізованих рішень для інтеграції з медичними базами даних і не забезпечують автоматичної верифікації та корекції розпізнаних даних з урахуванням специфічних вимог медичних аналізів.

# 3 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

У цьому розділі описано програмну реалізацію системи для формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображеннях. Розглянуто архітектуру системи та її шаблон проєктування. Спроєктовано функціонал системи та розглянуто модель представлення даних у базі даних.

## 3.1 Огляд архітектури системи

Система розроблена з використанням монолітної архітектури програмування. Цей підхід до розробки програмного забезпечення передбачає побудову програми як єдиного цілого, що містить усі його компоненти, що тісно взаємопов’язані та виконуються в рамках одного процесу. Монолітна архітектура визначає, що програма розгортається як єдине ціле, що потребує перебудови та повторного розгортання програми щоразу, коли вносяться будь-які зміни.

Типова структура програми в монолітній архітектурі зображена на рисунку 3.1 та передбачає поділ програми на три окремі рівні: рівень інтерфейсу (презентація), рівень контролеру (бізнес-логіки) та рівень моделі (зберігання даних). Ці шари складно взаємопов’язані, і зміни, внесені до одного шару, можуть мати наслідки для інших [15, 16, 17].

При використанні монолітної архітектури програмування зазвичай використовується шаблон проєктування програмного забезпечення Model-View-Controller (MVC). Розроблена система його також використовує.

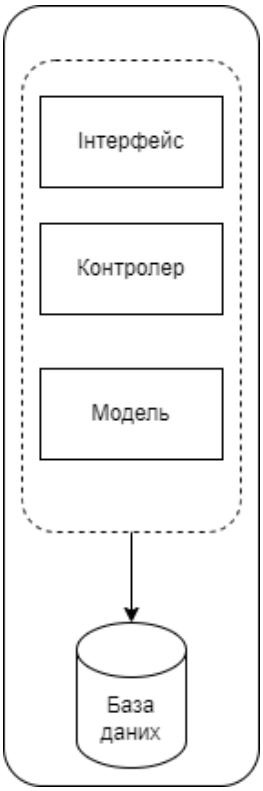


Рисунок 3.1 – Структура монолітної архітектури

Використання цього шаблону обумовлює розділення системи на три взаємопов’язані частини, щоб відокремити внутрішню логіку додатку від користувацького інтерфейсу, створюючи проміжний шар. MVC дозволяє розробникам логічно структурувати великі додатки, поділяючи їх на окремі частини, кожна з яких має своє призначення [18]. Діаграма системи MVC представлена на рисунку 3.2.

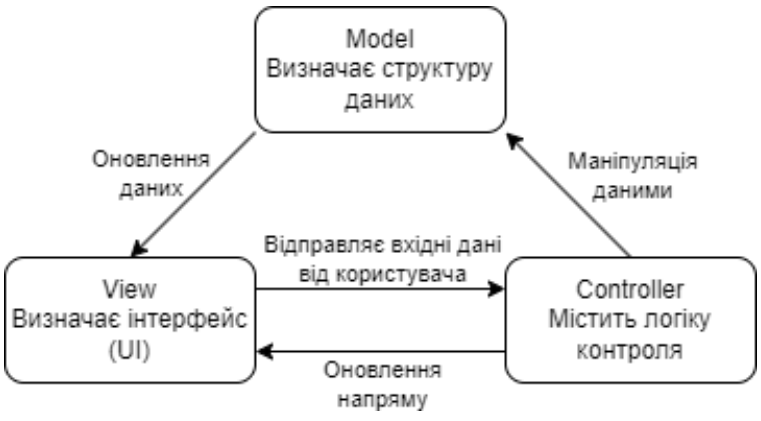


Рисунок 3.2 – Діаграма системи MVC

Шар Model – місце де знаходяться дані додатку. Цей шар містить реалізацію додатку про взаємодію з базою даних системи. Шар Model містить методи підключення до бази даних, також у цій частині відбуваються запити до бази даних та управління отриманими даними.

Шар View – містить інтерфейс через який користувачі взаємодіють з додатком. Цей шар містить усі файли HTML, при використанні архітектури MVC. Шар не може напряму взаємодіяти з шаром Model, для цього він використовує шар Controller.

Шар Controller – забезпечує зв’язок між шаром Model та View. Його можна уявити як офіціанта в ресторані, що обробляє замовлення користувачів ( запит з шару View ). Далі цей офіціант іде до кухні ( шар Model ), де він отримує замовлену їжу і подає її користувачу, що є обробкою запиту контролером.

### 3.2 Проєктування функціоналу системи

Для кращого розуміння функціоналу системи використано діаграму прецедентів (use case diagram) зображену на рисунку 3.3. Використання діаграми use case допомагає відобразити взаємодію між різними користувачами та функціоналом системи.

На діаграмі прецедентів визначено двох основних акторів: User та Admin. User представляє собою особу, що взаємодіє з системою для завантаження, перегляду та управління лише власними лабораторними аналізами. В той час як Admin може завантажувати, переглядати та управляти усіма завантаженими лабораторними аналізами.

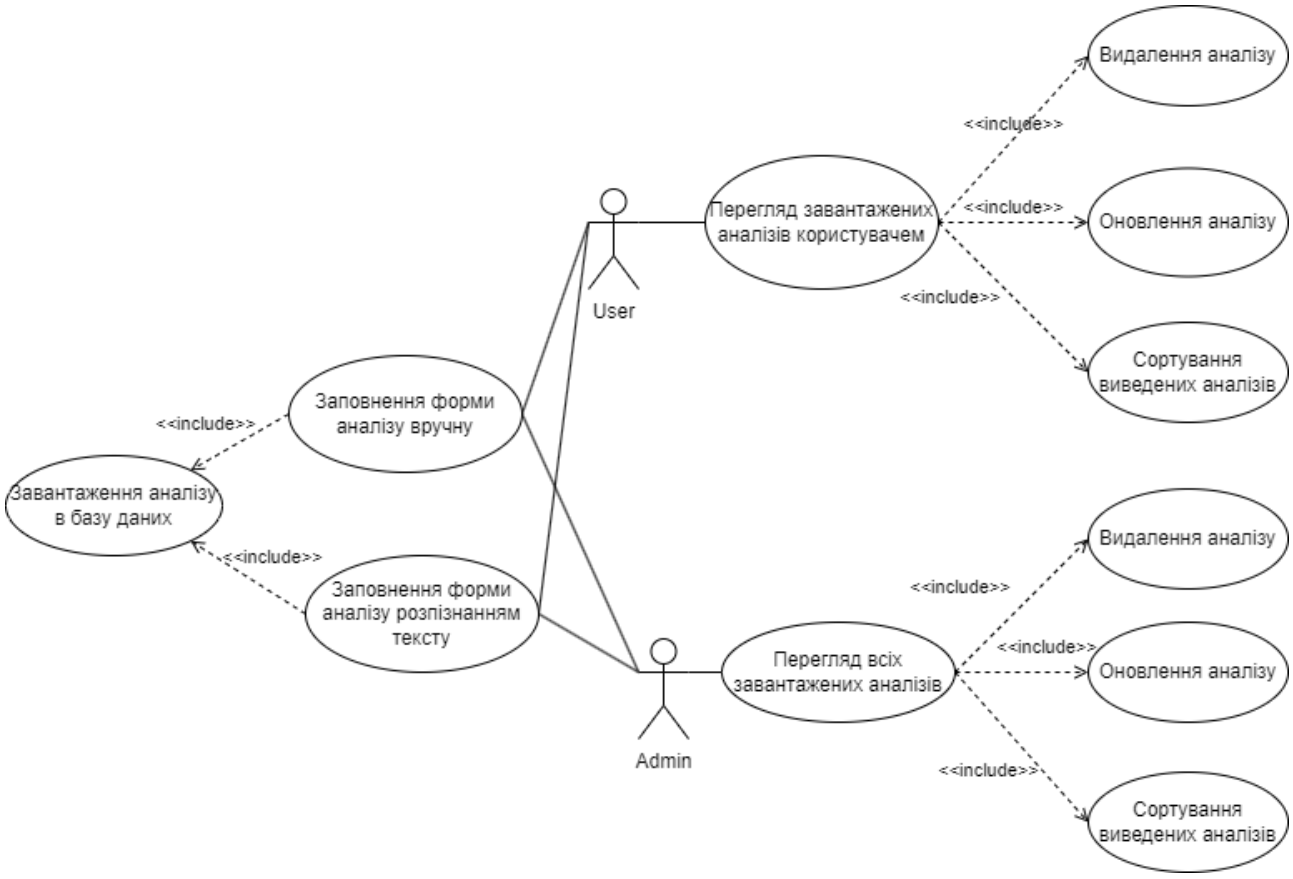


Рисунок 3.3 – Діаграма прецедентів додатку

Основні зображені прецеденти (функції) системи:

Завантаження аналізу в базу даних: в залежності від методу введення даних, вручну чи розпізнанням тексту на зображенні, лабораторний аналіз завантажується до бази даних під унікальним id.

Перегляд вмісту бази даних: в залежності від ролі користувача, user – бачить лише свої завантажені аналізи та admin – бачить усі завантажені аналізи в базі даних, надається перегляд даних та можливість видалення, оновлення, сортування отриманих лабораторних аналізів.

### 3.3 Модель представлення даних

Модель представлення даних є фундаментальним елементом будь-якої інформаційної системи, включаючи систему формування бази даних лабораторних

аналізів. Цей розділ описує структуру та організацію даних, які використовуються для зберігання та обробки інформації про користувачів та результати аналізів.

База даних представляє собою дві таблиці, з’єднані за допомогою первинного ключа. Модель представлення даних в базі даних зображена на рисунку 3.4.

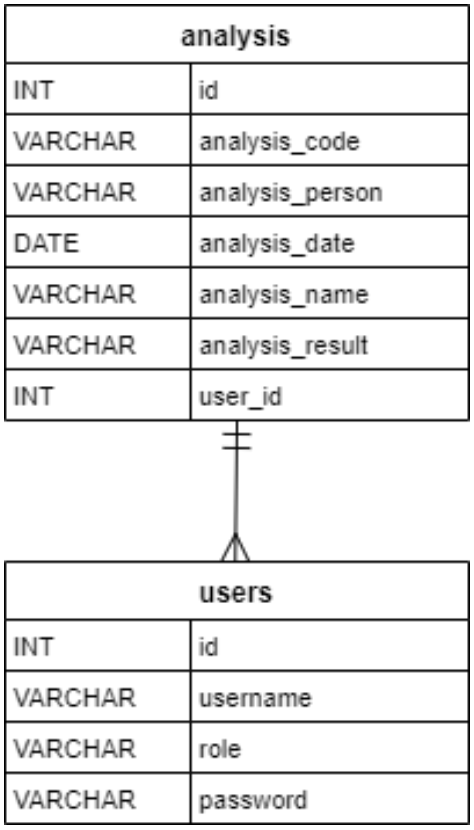


Рисунок 3.4 – Модель представлення даних в базі даних

Таблиця “users” використовується для зберігання інформації про користувачів системи, її застосування дозволяє системі виконувати процес реєстрації та автентифікації користувачів, керувати доступом до функціоналу програмного забезпечення. Вона містить поля id, username, role, password Поле id містить унікальний номер аккаунту користувача, це числове значення, яке автоматично збільшується при реєстрації нового користувача. Поле username містить інформацію про ім’я користувача, за яким він входить в систему. Це строкове значення, яке є унікальним. Поле role містить інформацію про роль користувача у системі, системою визначено дві ролі для користувачів – user та admin. Це поле має строкове значення. Поле password містить пароль користувача

у зашифрованому вигляді, забезпечуючи безпеку даних. Це поле строкового значення, що шифрується алгоритмом хешування перед завантаженням нового користувача.

Таблиця “analysis” використовується для зберігання інформації про проведені лабораторні аналізи. Вона містить поля id, analysis_code, analysis_person, analysis_date, analysis_name, analysis_result, user_id. Поле id містить унікальний номер аналізу, це числове значення, яке автоматично збільшується при завантажені нового аналізу. Поле analysis_code містить код аналізу, воно є строковим значенням. Поле analysis_person містить ПІБ пацієнта, власника аналізу. Це поле є строковим значенням. Поле analysis_date вказує на дату проведення аналізу, це поле має значення дати. Поле analysis_name є строковим та містить інформацію про назву аналізу, що був проведений. Поле analysis_result є строковим та має дані що характеризують результат аналізу. Поле user_id вказує на користувача, що завантажив аналіз до бази даних.  Це ідентифікатор користувача, який зв'язує цей запис з таблицею “users”.

Створена модель представлення даних дозволяє ефективно зберігати та управляти інформацією про користувачів та результати аналізів, що є ключовими компонентами системи формування бази даних лабораторних аналізів.

### 3.4 Опис основних модулів системи

Процес формування бази даних включає кілька ключових етапів: отримання зображень аналізів, попередня обробка зображень, розпізнавання тексту (OCR), верифікація та корекція даних, збереження в базі даних. Зображення можуть бути отримані за допомогою сканування паперових документів або безпосередньо з цифрових пристроїв. Цей розділ описує найголовніші модулі розробленої системи.

Розглядаючи ключові компоненти та структурні елементи, можна визначити основні модулі, що забезпечують функціональність системи, такі як модуль завантаження зображень, модуль розпізнавання тексту, модуль обробки та структурування отриманих даних, модуль роботи з базою даних, модуль

авторизації користувачів. Для реалізації цієї системи використовуються різні технології та інструменти, зокрема мова програмування JavaScript, OCR бібліотеки та сервіси (Tesseract, Google Cloud Vision), база даних PostgreSQL.

Модуль завантаження зображень відповідає за імпорт та зберігання зображень лабораторних аналізів для подальшого етапу розпізнавання тексту. Цей модуль без перебільшень є критично важливим для подальшого розпізнання тексту на завантажених зображеннях.

Модуль розпізнавання тексту є ключовим компонентом у системі. Використовуючи алгоритми машинного навчання та комп'ютерного зору, він здатний виявляти та розпізнавати текстову інформацію на зображеннях лабораторних аналізів з досить високою точністю.

Модуль обробки та структурування отриманих даних забезпечує перетворення розпізнаного тексту в структуровану інформацію, що вже може бути придатною для збереження в базі даних.

Модуль роботи з базою даних відповідає за надійне та ефективне зберігання, структурування та маніпулювання медичних даних, отриманих в результаті розпізнавання тексту на зображеннях лабораторних аналізів.

Модуль авторизації користувачів відіграє критично важливу роль у забезпеченні безпеки та конфіденційності завантажених медичних даних у системі від несанкціонованого доступу. Цей модуль відповідає за контролем доступу до системи розпізнання тексту та системи роботи з базою даних (лише авторизовані користувачі матимуть доступ до цих систем).

Архітектура системи для формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображеннях є комплексною та інтегрованою. Ця система передбачає процеси завантаження зображень, розпізнання тексту, роботи з базою даних, авторизацією користувачів, а також забезпечує зручний інтерфейс для роботи з системою.

### 3.4.1 Модуль завантаження зображень

Одне з завдань для подальшого розпізнавання тексту з зображень є можливість завантажувати файли на сервер. Для вирішення цього завдання в

Node.js використовується бібліотека Multer. Схема модуля зображена на рисунку 3.5.


Рисунок 3.5 – Схема модуля завантаження зображень

Multer — популярне проміжне програмне забезпечення для Node.js, яке керує процесом завантаження файлів на сервер у програмах Node.js. Ця бібліотека надає зручний спосіб керувати завантаженими файлами та зберігати їх на сервері чи хмарному сховищі [19].

Переваги використання Multer:

1. Простота використання: Multer має зрозумілий API, що значно спрощує процес завантаження файлів.

2. Гнучкість конфігурації: бібліотека надає різні параметри для збереження файлів на диску, такі як можливість налаштування шляху для збереження файлів, налаштування їх іменування, обмеження розміру.

3. Багатопотокове завантаження: бібліотека підтримує можливість одночасного завантаження як одного, так і декількох файлів.

4. Інтеграція з Express.js: Multer зазвичай використовується як проміжне програмне забезпечення (middleware) у додатках з фреймворком Express.js. Його можна додавати до маршрутів для обробки завантажених файлів, після чого можна отримати доступ до них в об’єкті запиту для подальшої роботи.

5. Різні механізми зберігання: Multer має підтримку зберігання файлів на диску пристрою, в пам’яті хмарного сховища, наприклад, Amazon S3.

6. Валідація файлів: бібліотека має змогу перевіряти завантажені файли за їх типом, розміром та іншими критеріями. Це забезпечує обробку лише потрібних файлів.

### 3.4.2 Модуль розпізнання тексту на зображені

Модуль відповідає за розпізнавання тексту з завантажених зображень. В цьому модулі використовується дві потужні бібліотеки розпізнавання тексту: Tesseract.js та Google Cloud Vision. Схема модуля зображена на рисунку 3.6.


Рисунок 3.6 – Схема модуля розпізнання тексту на зображені

Бібліотека Tesseract.js є JavaScript портом популярного рушія оптичного розпізнавання символів (OCR) Tesseract. Бібліотека Tesseract створена компанією Hewlett-Packard у 80-х та 90-х роках минулого століття. Портування на операційну систему Windows відбулося у 1996 році, у 1998 році програму переписали на мову програмування C++. У 2005 році компанія HP оприлюднила вихідний код програми. Після цього подальший розвиток Tesseract здійснювала компанія Google протягом більше ніж десяти років до 2018 року. Подальший розвиток бібліотеки продовжується групою розробників з усього світу на волонтерських засадах. Станом на сьогодні Tesseract є проєктом з відкритим кодом, основний репозиторій якого розміщений на GitHub [11, 20].

Для розпізнавання тексту ця бібліотека використовує гібридний підхід, який поєднує різні методи розпізнавання:

1. Виділення ознак символів (feature extraction).
2. Двовимірний метод згортки (2D convolutional approach).

3. Рекурсивні нейронні мережі довгої короткочасної пам'яті (LSTM).

Спочатку Tesseract виділяє лінії тексту, розбиває їх на слова, а потім окремі символи. Для розпізнавання символів використовуються згорткові нейронні мережі та рекурентні нейронні мережі LSTM. Це забезпечує високу точність для різних типів символів та шрифтів.

Використання бібліотеки є повністю безкоштовним, для розпізнавання тексту використовуються обчислювальна потужність системи, тому швидкість розпізнавання є різною в залежності від комплектуючих комп’ютера. Використання бібліотеки Tesserct спеціалізується на розпізнанні лише друкованого тексту. Для покращення результату розпізнання тексту необхідно використовувати зображення високої якості.

Бібліотека Google Cloud Vision – є потужним інструментом для розпізнавання тексту на зображеннях. Розробкою займається компанія Google. Компанія Google виконала складну роботу з навчання моделей комп'ютерного зору на величезній кількості зображень, використовуючи потужні обчислювальні ресурси Google. Створено бібліотеки для різних мов програмування, таких як: C#, Java, JavaScript, Python. Потрібно просто надіслати Google зображення та обрати завдання, наприклад, виявлення об'єктів чи тесту. Далі Google обробляє запит та надсилає результати назад. Отже, можна легко інтегрувати функції комп'ютерного зору у свою програму [12, 21].

Для розпізнавання тексту Google Cloud Vision використовує глибоке навчання з передовими нейронними мережами. Точна архітектура нейронних мереж не розголошується, але відомо, що вона базується на згорткових та рекурентних нейронних мережах.

Крім того, Google Cloud Vision використовує додаткові методи обробки для поліпшення якості розпізнавання, такі як корекція нахилу тексту, усунення шумів та обробка складних фонів.

Використання бібліотеки не є повністю безкоштовним, користувачу надається ліміт безкоштовних використань на місяць, детальніше про ціни можна дізнатися з таблиці ціноутворення зображеної на рисунку 3.7.

При розробці модуля розпізнання тексту з зображень використовується бібліотека для JavaScript та метод Text Detection. Розпізнання тексту відбувається на сервері Google тому швидкість розпізнання є високою, необхідний лише доступ до інтернету. Використання бібліотеки Google Cloud Vision спеціалізується на розпізнанні рукописного тексту та суміші з друкованого та рукописного тексту.

| Feature | Price per 1000 units | | |
|---|---|---|---|
| | First 1000 units/month | Units 1001 - 5,000,000 / month | Units 5,000,001 and higher / month |
| Label Detection | Free | $1.50 | $1.00 |
| Text Detection | Free | $1.50 | $0.60 |
| Document Text Detection | Free | $1.50 | $0.60 |
| Safe Search (explicit content) Detection | Free | Free with Label Detection, or $1.50 | Free with Label Detection, or $0.60 |
| Facial Detection | Free | $1.50 | $0.60 |
| Facial Detection - Celebrity Recognition | Free | $1.50 | $0.60 |
| Landmark Detection | Free | $1.50 | $0.60 |
| Logo Detection | Free | $1.50 | $0.60 |
| Image Properties | Free | $1.50 | $0.60 |
| Crop Hints | Free | Free with Image Properties, or $1.50 | Free with Image Properties, or $0.60 |
| Web Detection | Free | $3.50 | Contact Google for more information |
| Object Localization | Free | $2.25 | $1.50 |

Рисунок 3.7 – Ціноутворення Google Cloud Vision

Остаточне порівняння використаних бібліотек для розпізнавання тексту зображене в таблиці 1.

Таблиця 1 – Порівняння Tesseract.js та Google Cloud Vision

| | Tesseract.js | Google Cloud Vision |
|---|---|---|
| Підтримка великої кількості мов | + | + |

Продовження таблиці 1

| | | |
|---|---|---|
| Підтримка розпізнання друкованого тексту | + | + |
| Підтримка розпізнання рукописного тексту | - | + |
| Залежність від обчислювальної потужності комп’ютеру | + | - |
| Залежність від зв’язку з інтернетом | - | + |
| Ціна | + (безкоштовно) | +- (надається лише 1000 безкоштовних використань на місяць) |

Модуль розпізнавання тексту забезпечує вибір між цими двома бібліотеками залежно від вимог до швидкості, точності та доступних обчислювальних ресурсів. При використанні додатку ми можемо використовувати Tesseract.js для локального розпізнавання тексту друкованого тексту або використовувати Google Cloud Vision для більш потужних хмарних обчислень, наприклад, зображення рукописного тексту або суміш друкованого тексту та рукописного.

### 3.4.3 Модуль обробки та структурування отриманих даних

Модуль відповідає за обробку та структурування отриманих даних від модуля розпізнавання тексту. Він забезпечує перетворення вхідних даних в придатну форму для подальшої зручної роботи. Модуль використовує регулярні вирази для обробки даних медичних досліджень за шаблоном. Схема модуля зображена на рисунку 3.8.


Рисунок 3.8 – Схема модуля обробки та структурування даних

Регулярні вирази – це спосіб описання шаблону в текстових рядках. Використання регулярних виразів є потужним інструментом для перевірки та обробки текстових рядків. В JavaScript застосування регулярних виразів є не дуже зручним, оскільки JavaScript не надає зручного інструменту для простого їх написання [22].

Модуль перевіряє правильність написання текстового рядку з модуля розпізнавання тексту за шаблоном, якщо перевірка проходить успішно модуль розбиває початковий текстовий рядок на окремі рядки, які можна використовувати у модулі роботи з базою даних, якщо початковий рядок не проходить перевірку – то користувачу необхідно вручну відредагувати текстовий рядок за шаблоном, зображеним на рисунку 3.9.

Дослідження: 11111
ПІБ: І'мя Прізвище
Дата: 06.10.2000
Назва дослідження: аналіз крові
Результат:
Гемоглобін 123
...

Рисунок 3.9 – Шаблон лабораторного дослідження

### 3.4.4 Модуль роботи з базою даних

Модуль відповідає за встановлення з’єднання з базою даних, виконання запитів до бази даних. Схема модуля зображена на рисунку 3.10. Модуль використовує систему управління базою даних (СУБД) PostgreSQL.


Рисунок 3.10 – Схема модуля роботи з базою даних

PostgreSQL — це потужна об’єктно-реляційна система керування базами даних з відкритим кодом. Вона використовує мову SQL і поєднує її з можливостями безпечного зберігання даних, щоб витримувати складні робочі навантаження. Початки PostgreSQL сягають 1986 року в рамках проєкту POSTGRES в Каліфорнійському університеті Берклі, і вона активно розроблявся більше 35 років [23].

СУБД PostgreSQL має чудову репутацію та стала провідною реляційною базою даних із відкритим вихідним кодом для багатьох окремих розробників та організацій.

Модуль виконує такі функції з базою даних PostgreSQL:

1. Зберігання даних лабораторних аналізів.
2. Видалення лабораторного аналізу.
3. Редагування лабораторного аналізу.
4. Вивід даних з бази даних.

### 3.4.5 Модуль авторизації користувачів

Модуль авторизації користувачів відповідає за процес автентифікації та надання доступу до системи. Цей модуль є важливою частиною системи, оскільки

він забезпечує захист самої системи, а також захищає конфіденційні дані, лабораторних аналізів, від зловмисників. Модуль передбачає, що лише авторизовані користувачі можуть отримати доступ до функцій додатку. Схема модуля зображена на рисунку 3.11.


Рисунок 3.11 – Схема модуля авторизації користувачів

Після реєстрації логін та пароль користувача зберігається в базі даних. Для покращення захисту цих даних, зокрема паролю, використовується бібліотека bcrypt. Далі коли зареєстрований користувач успішно вводить свої логін та пароль сервер генерує йому Json Web Token (JWT) куди записується його id та роль та час життя цього токену, після чого токен завантажується в кукі в браузері та користувачу не потрібно буде знову авторизуватися на сайті протягом часу коли токен вважається дійсним або поки користувач не захоче вийти зі свого аккаунту.

Bcrypt - криптографічна хеш-функція, створена у 1999 році Нільсом Провосом та Девідом Мазьєром, використовуючи за базу алгоритм шифрування Blowfish. Він призначений для хешування паролів за допомогою методу, який не є вразливим до атак на основі словника для безпечного зберігання на стороні сервера.

Bcrypt використовує функцію одностороннього хешування, тому після хешування пароля його неможливо повернути до початкової форми. Для того щоб захисти пароль ще краще, bcrypt додає випадкові дані – “сіль” щоб створити унікальний хеш, після цієї дії пароль стає майже неможливо взламати автоматизованими способами повного перебору. Після хешування пароля він

перетворюється на рядок фіксованої довжини. Щоразу, коли користувач намагається увійти до свого облікового запису, bcrypt хешує введений пароль і порівнює його з версією, що зберігається на сервері, щоб перевірити, чи збігається пароль.

Bcrypt виділяється серед інших алгоритмів хешування використанням фактора вартості. Ця функція дозволяє визначати кількість ітерацій пароля та раундів хешування, що збільшує час і обчислювальні ресурси, необхідні системі для обчислення остаточного хешованого пароля. Це перетворює bcrypt на повільний алгоритм, тому що потрібно більше часу на генерацію ключа хешування, але значно збільшує захист паролів [24].

JSON Web Token (JWT) – це JSON об’єкт, стандартизований в RFC 7519. Використовується переважно для авторизації користувачів та подальшого дозволу використання користувачами функціоналу системи.

Структура JSON Web Token складається з трьох частин, розділених крапкою:

1. Заголовок header. Ця частина складається з двох частин: тип токену, який є JWT, та алгоритм хешування даних, що використовуватиметься при створенні підпису. В розробленій системі буде використовуватися алгоритм HS256, що використовує секретний ключ – текстовий рядок.

2. Дані збережені в токені payload. В цю частину записується id користувача та його роль.

3. Підпис signature. Ця частина отримується в результаті кодування header та payload алгоритмом base64url, після чого алгоритм об’єднує отримані два закодованих рядки в один через крапку, отриманий рядок хешується алгоритмом, що був заданий в header на основі секретного ключа.

# 4 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

У цьому розділі описано роботу користувача з програмною системою для формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображеннях. Наведено інструкції щодо встановлення та налаштування системи. Продемонстровано функціонал системи для користувача. Представлено результати тестування системи на наборі тестових зображень.

## 4.1 Інсталяція програмного забезпечення

Розроблене програмне забезпечення створювалося та тестувалося на такій системі:

- Операційна система: Windows 10 (64-bit)
- Процесор: AMD Ryzen 5 5600
- Відеокарта: Nvidia RTX 4060
- Оперативна пам’ять: 32 гб ddr4 3200гц
- Накопичувач: SSD 1 тб

Розробка програмного забезпечення відбувалася у середовищі Visual Studio Code. Необхідні залежності описуються в файлі package.json. Версії бібліотек, що використовувалися при розробці програмного забезпечення:

1. @google-cloud/vision": "^4.2.1",
2. bcrypt: 5.1.1,
3. body-parser: 1.20.2,
4. cookie-parser: 1.4.6,
5. ejs: 3.1.10,
6. express: 4.19.2,
7. jsonwebtoken: 9.0.2,
8. multer: 1.4.5-lts.1,

9. pg: 8.11.5,
10. tesseract.js: 5.0.5.

Для встановлення потрібних бібліотек потрібно завантажити середовище Node.js це можна зробити з офіційного сайту [25]. Разом з ним встановлюється пакетний менеджер npm, що використовується для встановлення бібліотек. Для встановлення залежностей в терміналі потрібно запустити команду “npm install”, після виконання в терміналі виведеться повідомлення про завантаження пакетів зображене на рисунку 4.1.

```
PS D:\project> npm install

added 290 packages, and audited 291 packages in 3s

27 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 4.1 – Завантаження пакетів

Для запуску серверу потрібно виконати команду в терміналі “npm start”, успішним запуском серверу є вивід повідомлення у терміналі, що зображено на рисунку 4.2.

```
PS D:\project> npm start

> project@1.0.0 start
> nodemon server.js

[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node server.js`
Server started: http://localhost:3000
```

Рисунок 4.2 – Успішний запуск серверу

## 4.2 Демонстрація функціоналу програми

### 4.2.1 Опис головної сторінки

Усі користувачі незалежно від ролей спочатку потрапляють на головну сторінку сайту системи зображену на рисунку 4.3, якщо не авторизований користувач спробує перейти на інше посилання системи вручну його автоматично направляє на головну сторінку.


Рисунок 4.3 – Головна сторінка системи

### 4.2.2 Опис сторінки реєстрації/входу

Після натискання кнопки Реєстрація/Вхід користувач потрапляє на сторінку створення аккаунту та входу до нього. Вхід користувача зображено на рисунку 4.4.

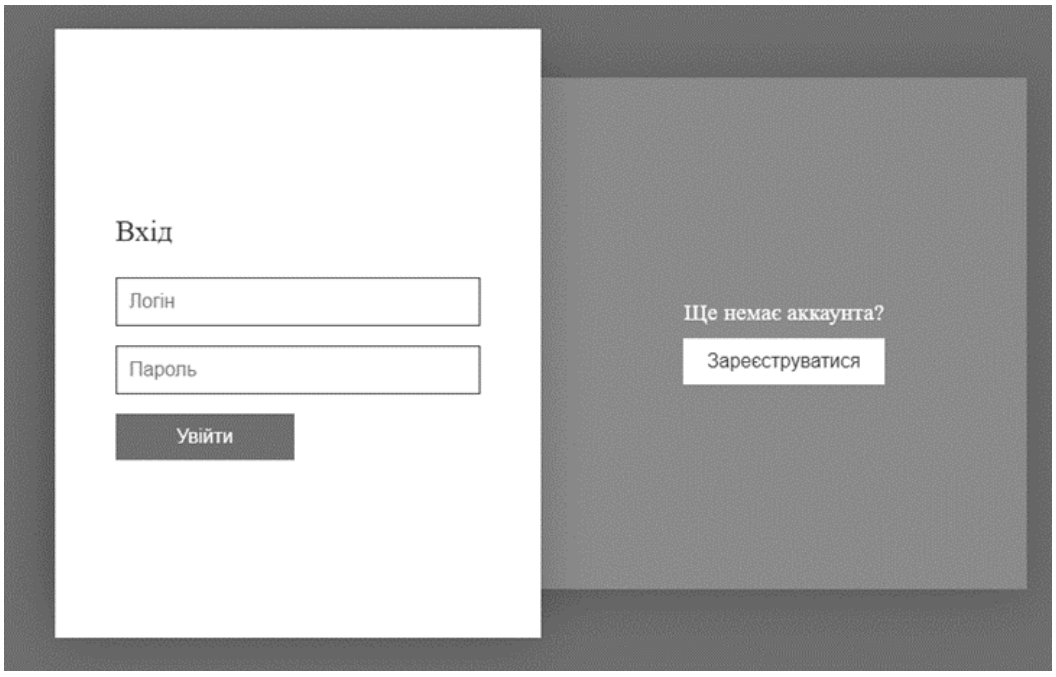


Рисунок 4.4 – Вхід до аккаунта

Реєстрація користувача зображена на рисунку 4.5.

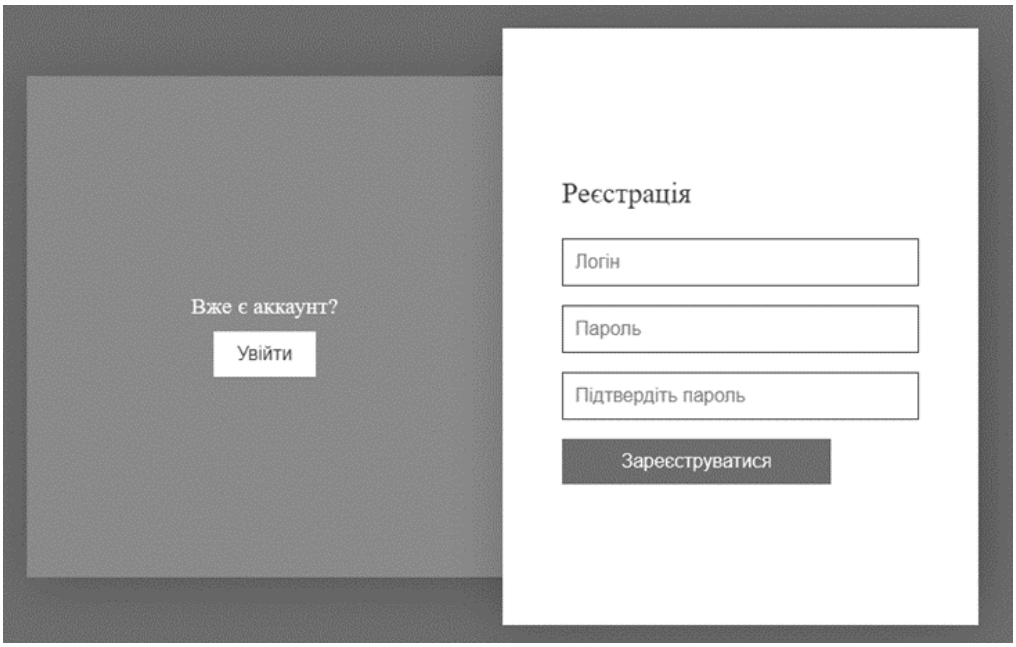


Рисунок 4.5 – Реєстрація аккаунта

Після успішної авторизації у системі користувач потрапляє на головну сторінку системи, де вже для нього надається доступ до основновного функціоналу

системи зображеного на рисунку 4.6. Він може вийти в аккаунту, перейти на сторінку з даними з бази даних, перейти на сторінку розпізнання тексту (вводу даних до бази даних).


Рисунок 4.6 – Користувач авторизувався у системі

### 4.2.3 Опис сторінки розпізнання тексту

Сторінка розпізнавання тексту представляє собою вибір трьох варіантів заповнення даними бази даних. При виборі методу Tesseract користувач отримує можливість завантаження зображення та при натисканні кнопки “розпізнати текст” запускає алгоритм розпізнання тексту на зображені Tesseract.js, зображено на рисунку 4.7.

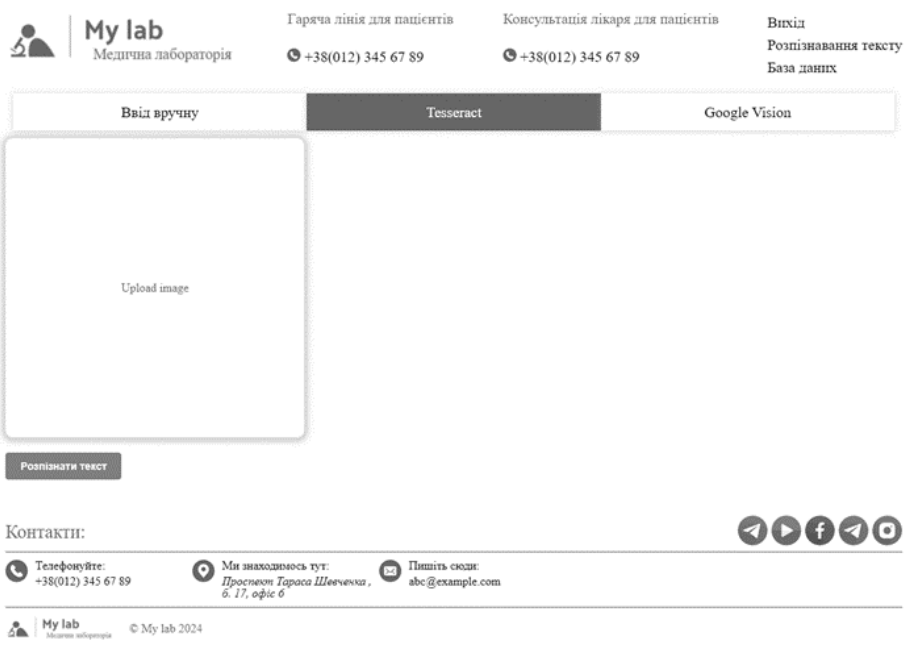


Рисунок 4.7 – Вибір методу Tesseract

При виборі методу “Google Vision” користувач аналогічно попередньому методу отримує можливість завантаження зображення та при натисканні кнопки “розпізнати текст” запускає алгоритм розпізнання тексту на зображені Google Vision API text detection, зображено на рисунку 4.8.

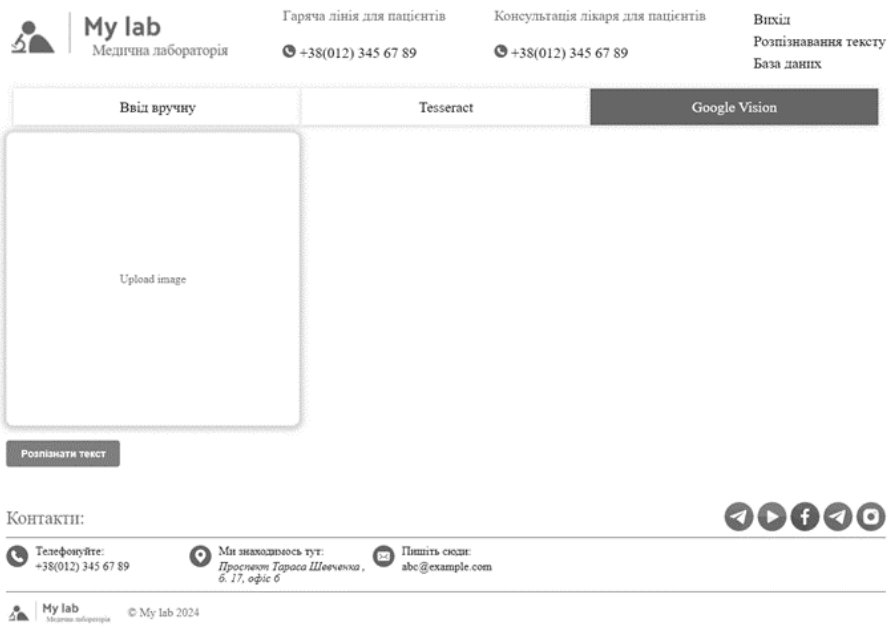


Рисунок 4.8 – Вибір методу Google Vision

При виборі методу “Ввід вручну” користувач отримує можливість заповнення форми про лабораторний аналіз вручну, що зображено на рисунку 4.9.

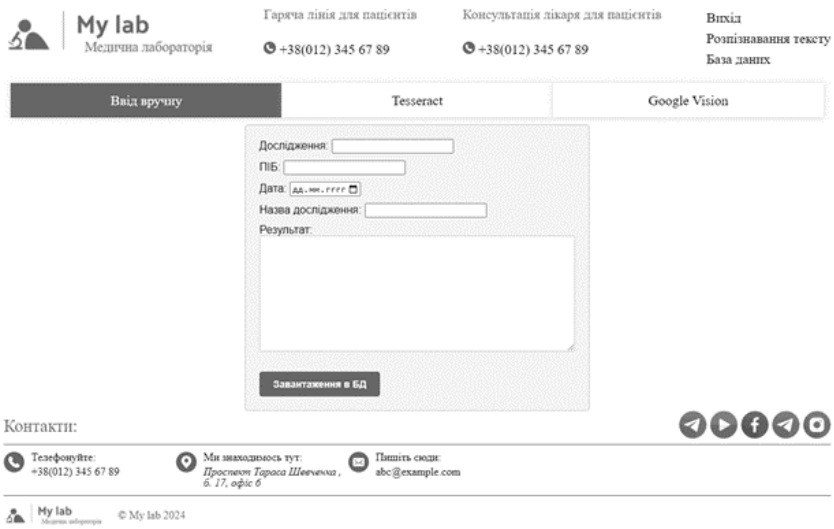


Рисунок 4.9 - Вибір методу Ввід вручну

Розглянемо функціонал при виборі методів розпізнання тексту детальніше. Після завантаження бажаного зображення користувач може його переглянути або завантажити інше зображення, зображено на рисунку 4.10.


Рисунок 4.10 – Користувач завантажив зображення

Якщо розпізнавання тексту пройшло успішно та система змогла класифікувати отримані дані, виводиться заповнена форма аналізу зображена на рисунку 4.11, де користувач може відредагувати дані перед завантаженням до бази даних.

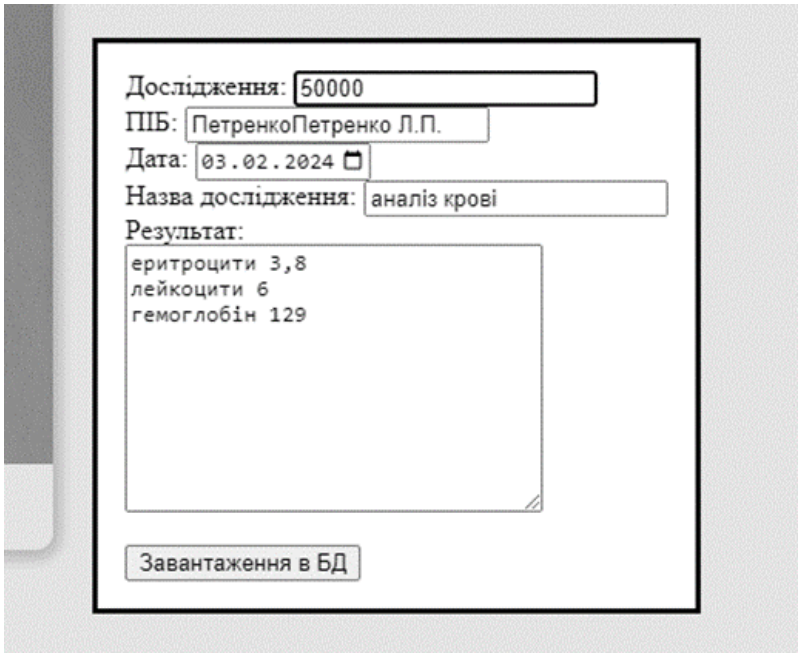


Рисунок 4.11 – Вивід заповненої форми аналізу

Якщо збереження до бази даних відбулося успішно, користувачу виводиться повідомлення зображене на рисунку 4.12, де йдеться про успішне завантаження та під яким id завантажено аналіз.


Рисунок 4.12 – Успішне завантаження аналізу до бази даних

Якщо розпізнавання тексту пройшло успішно, але система не змогла класифікувати отримані дані – то виводиться повідомлення про помилку зображене на рисунку 4.13.

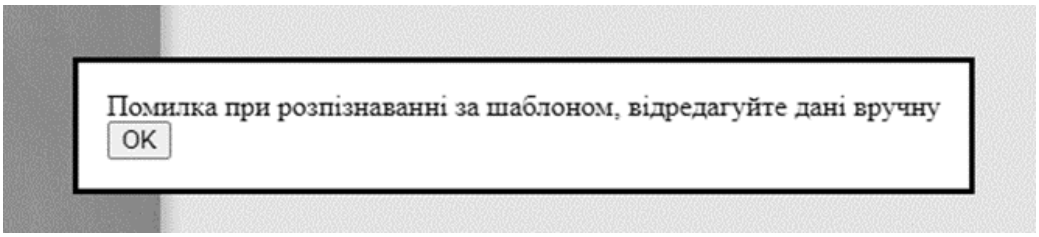


Рисунок 4.13 – Помилка при розпізнаванні зображення

Користувачу виводиться форма з текстом - результатом розпізнавання тексту, а також інструкцією як правильно відредагувати дані зображеною на рисунку 4.14.

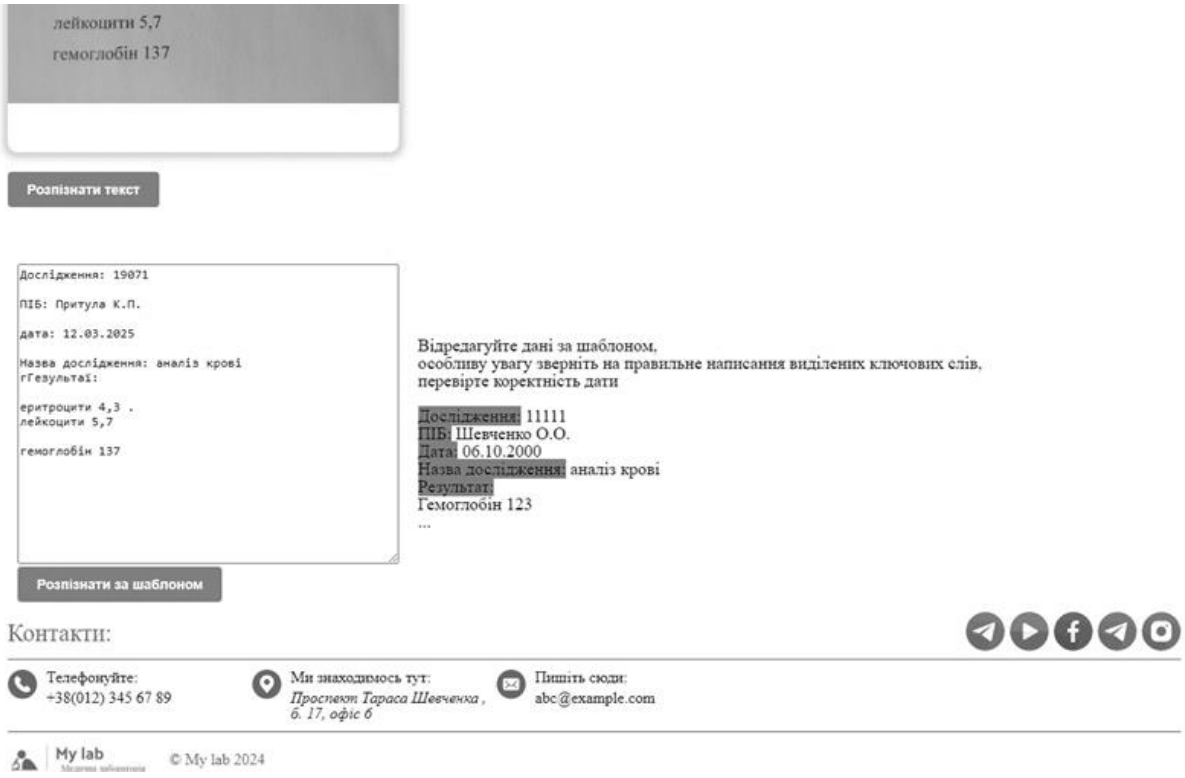


Рисунок 4.14 – Форма для виправлення помилок розпізнавання

У цьому прикладі виникла помилка у записі ключових слів “Дата:” та “Результат:”, виправлення помилки зображено на рисунку 4.15.

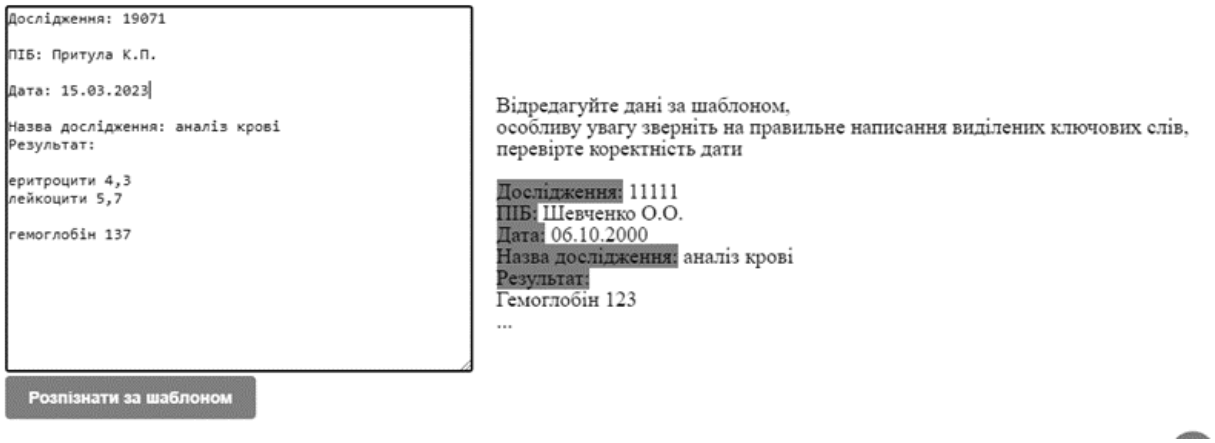


Рисунок 4.15 – Помилки розпізнавання виправлено

Після цього коли користувач натискає кнопку “Розпізнати за шаблоном” система знову намагатиметься класифікувати дані. При успіху виводиться форма зображена на рисунку 4.16, форма для виправлення помилок ховається. Кожного разу при невдалій спробі класифікації виводитиметься знову повідомлення про помилку з проханням відредагувати дані.

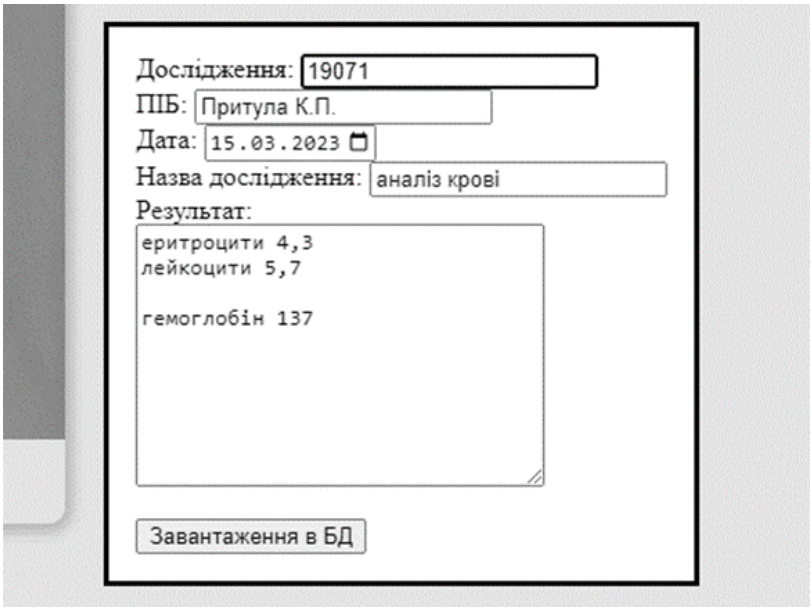


Рисунок 4.16 – Форма для вводу даних

### 4.2.4 Опис сторінки бази даних

Сторінка бази даних представляє собою вивід id користувача та його ролі, вивід даних з таблиці лабораторних аналізів, а також кнопки “Видалити” та “Змінити”, що є не активними поки не буде вибрано рядок таблиці з даними. Користувач який щойно зареєструвався отримує роль “user” та не може бачити завантажені аналізи інших користувачів, на рисунку 4.17 зображено вивід бази даних для користувача “user” що ще не додав жодного аналізу до бази даних.

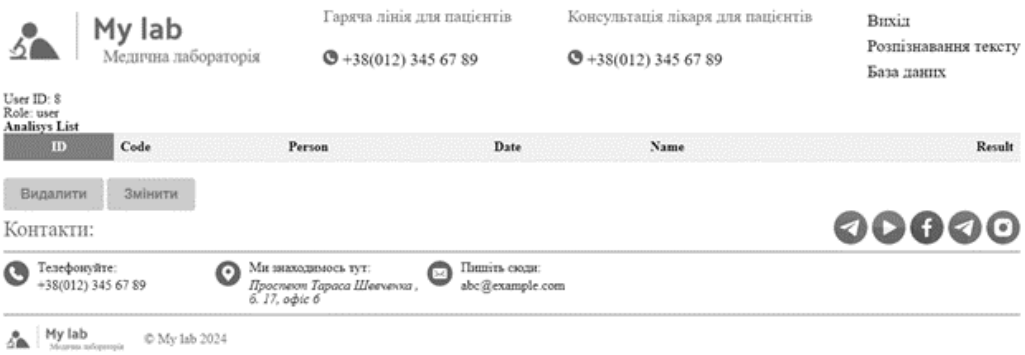


Рисунок 4.17 – Вивід бази даних нового користувача

Наш користувач завантажив декілька аналізів до бази даних. На рисунку 4.18 зображено вивід з бази даних його аналізів.

User ID: 8
Role: user
Analisys List

| ID | Code | Person | Date | Name | Result |
|---|---|---|---|---|---|
| 61 | 11111 | Шевченко А.О. | 10.05.2024 | аналіз крові | еритроцити 4,5 лейкоцити 6,2 гемоглобін 128 |
| 62 | 37394 | Романенко Л.В. | 13.02.2023 | ліпідний профіль | холестерин 5,5 тригліцериди 2,7 |
| 63 | 58672 | Притула к.п. | 15.08.2023 | рівень заліза | залізо 7,6 феритин 14,8 трансферитин 2,6 |
| 64 | 68034 | Романенко Л.В. | 16.08.2023 | рівень заліза | залізо 7,7 феритин 14,3 трансферитин 3,4 |
| 65 | 50000 | Петренко Л.П. | 03.02.2024 | аналіз крові | еритроцити 3,8 лейкоцити 6 гемоглобін 129 |
| 66 | 49724 | Коваленко М.С. | 14.03.2023 | аналіз крові | еритроцити 4,3 лейкоцити 5,9 гемоглобін 135 |
| 67 | 2004 | Бойко В.А. | 27.10.2023 | ліпідний профіль | холестерин 5,4 тригліцериди 2,2 |
| 68 | 50004 | Данильчук К.О. | 15.01.2024 | рівень заліза | залізо 6,6 феритин ли, ч 14,4 трансферитин 3,4 |
| 69 | 11111 | Шевченко А.О. | 10.05.2024 | аналіз крові | еритроцити 4,5 лейкоцити 6,2 гемоглобін 128 |
| 70 | 32166 | Коваленко М.С. | 24.03.2023 | аналіз крові | еритроцити 3,6 лейкоцити 5,8 гемоглобін 145 |

Далі

Видалити Змінити

Рисунок 4.18 – Приклад заповненої бази даних користувачем

Користувач з роллю “admin” бачить усі завантажені аналізи в базі даних, що зображено на рисунку 4.19.


Рисунок 4.19 – Вивід бази даних для адміна

В системі реалізовано, що при натисканні на заголовок стовпчика таблиці відбувається автоматичне сортування даних за цим стовпчиком. Натискання на рядок з даними робить кнопки “Видалити” та “Змінити” активними. Вибір стовпчика та рядка таблиці зображено на рисунку 4.20.

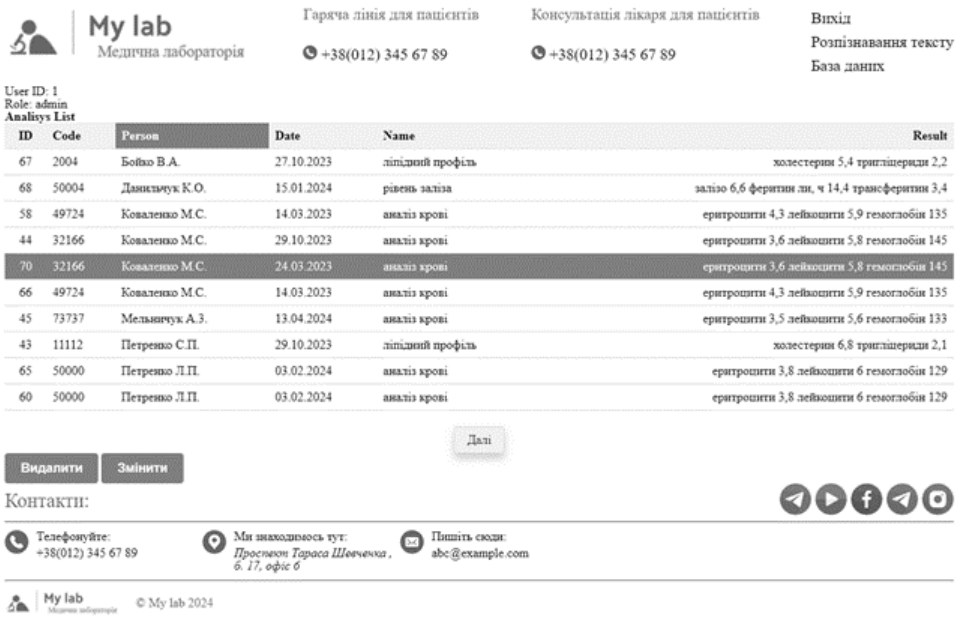


Рисунок 4.20 – Вибір стовпчика та рядка таблиці

При натисканні на кнопку “Видалити” видаляється виділений рядок з бази даних. При натисканні кнопки “Змінити” відкривається форма зображена на рисунку 4.21 для редагування виділеного аналізу.

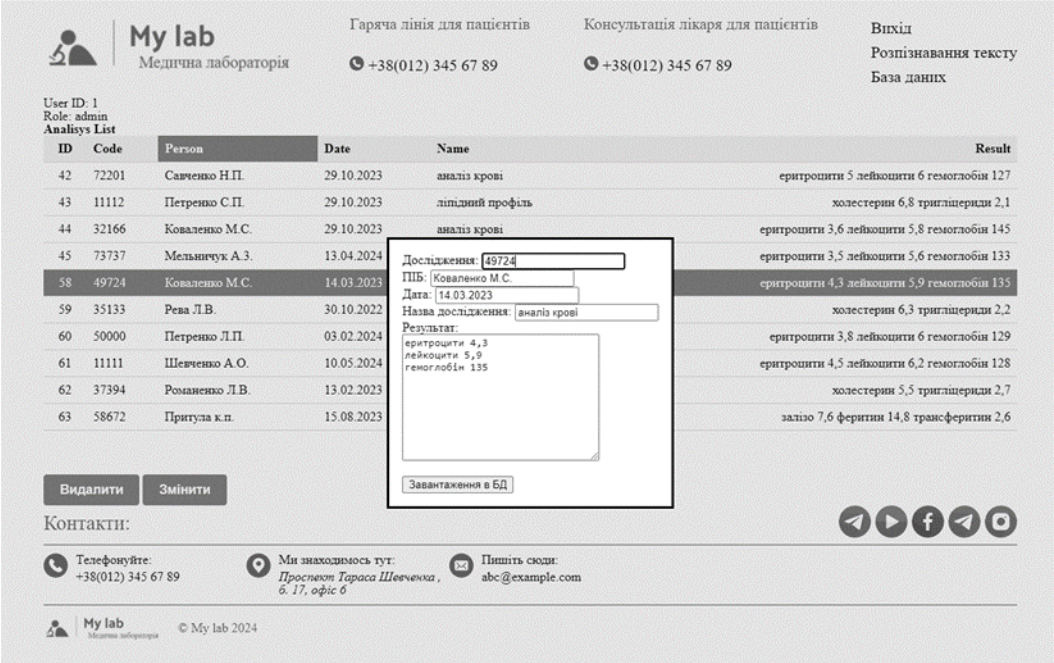


Рисунок 4.21 – Форма зміни даних аналізу

Після натискання кнопки “Завантажити в БД” оновлений аналіз завантажується в базу даних та сторінка перезавантажується для відображенння актуальних даних.

## 4.3 Оцінка точності обраних методів розпізнання тексту

### 4.3.1 Оцінка точності бібліотеки Google Vision API

Оцінка точності розпізнання тексту для бібліотеки Google Vision API відбувалася на 72 зображеннях. Результат оцінки точності 96%. Приклад одного з зображень для тестування зображено на рисунку 4.19.

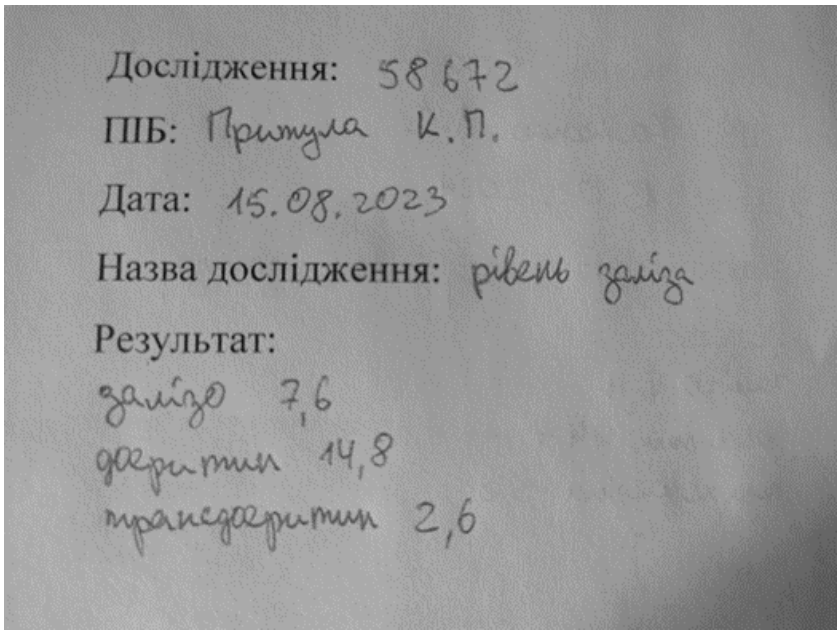


Рисунок 4.19 – Приклад зображення для оцінки бібліотеки Gogle Vision API

Основні помилки: плутання розміщення блоків тексту, плутання крапки та коми в полі з датою, плутання цифри 1 з цифрою 4, плутання цифри 4 з літерою ч, плутання літери З цифрою 3.

### 4.3.2 Оцінка точності бібліотеки Tesseract.js

Оцінка точності розпізнання тексту для бібліотеки Tesseract.js відбувалася на 11 зображеннях. Результат оцінки точності 95%. Приклад одного з зображень для тестування зображено на рисунку 4.20.

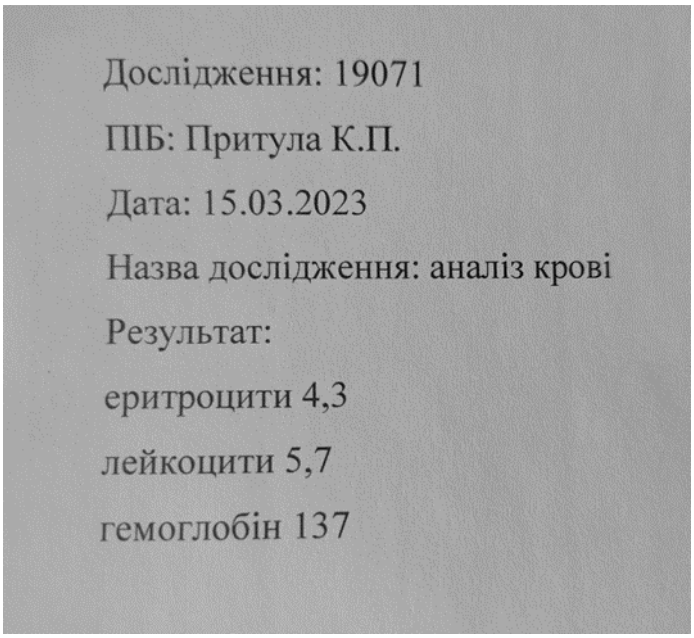


Рисунок 4.20 – Приклад зображення для оцінки бібліотеки Tesseract.js

Основні помилки: неправильне розпізнавання блоків тексту, плутання цифр, спроби розпізнавання тексту якого не має.

# ВИСНОВКИ

Дипломна робота присвячена розробці системи для формування бази даних лабораторних аналізів шляхом розпізнавання тексту на зображеннях. Основною проблемою створення бази даних лабораторних аналізів є складність збору та систематизації даних при виконанні цього вручну. Ручне введення даних є трудомістким та неефективним процесом схильним до допущення помилок, особливо при роботі з великими обсягами лабораторних аналізів. Використання технологій розпізнавання тексту на зображеннях мінімізує кількість допущених людських помилок, забезпечуючи вищу точність даних, та значно пришвидшує процес збору та систематизації даних.

В дипломній роботі розглянуто існуючі методи та алгоритми розпізнавання тексту на зображеннях такі, як метод шаблонів, метод ознак, штучні нейронні мережі, гібридні методи. Оглянуто існуючі системи розпізнавання тексту. На основі аналізу цих систем зроблено висновок, що розроблена система відрізняється від наявних рішень тим, що вона включає повний цикл обробки лабораторних аналізів: від попередньої обробки зображень і розпізнавання тексту до верифікації та автоматичного структурованого збереження даних у базі даних.

У рамках дослідження обрано для використання у системі методи розпізнання тексту на зображеннях на прикладі двох популярних бібліотек Tesseract.js та Google Vision API. При порівняні цих бібліотек розглядалися такі аспекти: підтримка розпізнання друкованого та рукописного тексту, точність розпізнавання, швидкість обробки, а також вартість доступу.

Розроблено алгоритм для структурування даних з розпізнаного тексту з застосуванням регулярних виразів. Цей алгоритм передбачає визначення шаблону даних, що підлягає витягуванню з тексту: номер дослідження, ПІБ, дати, назви дослідження та його результат. Далі алгоритм застосовує цей регулярний вираз до тексту, використовуючи функції пошуку за допомогою регулярних виразів, і зберігає знайдені збіги у відповідних структурах даних. Ці дані можна використовувати для подальшого збереження в базі даних.

Створено базу даних для зберігання результатів лабораторних аналізів та даних аккаунтів користувачів. Ця база даних забезпечує централізоване сховище для структурованого зберігання та організації даних, пов'язаних з лабораторними аналізами та обліковими записами користувачів. Вона дозволяє ефективно зберігати результати різноманітних лабораторних тестів та аналізів, включаючи дані про номер дослідження, ПІБ, дати, назви дослідження та його результат. Крім того, база даних містить інформацію про облікові записи користувачів, такі як ім'я користувача, його роль та пароль. Структура бази даних ретельно спроєктована для забезпечення належного зв'язку між таблицями результатів аналізів та таблицями користувачів, що полегшує пошук, фільтрацію та аналіз даних.

Результатом роботи є програмне забезпечення, де компоненти інтегровано в єдину систему. Розроблена програмна система містить інтерфейс для роботи з користувачем, сервер на якому виконується основна логіка системи, базу даних.

Розроблене програмне забезпечення протестовано на наборі тестових зображень, що дозволило оцінити точність та ефективність обраних двох бібліотек. Результати тестування показали, що обидві бібліотеки мають високу точність розпізнання тексту, а точніше Google Cloud Vison 96% та Tesseract.js 95%. На основі тестування доведено коректність роботи розробленого програмного забезпечення.

Додатково у роботі розглянуто питання ризиків та вразливостей зберігання конфіденційної інформації про результати аналізів пацієнтів у базі даних. На основі цього в програмне забезпечення впроваджено заходи безпеки від несанкціонованого доступу, такі як авторизація користувачів та контроль їх ролей.

Розроблене програмне забезпечення можна вдосконалити покращенням алгоритмів розпізнавання та розришенням функціоналу систем – інтеграцією з електронними медичними картками, забезпеченням працездатності з різними форматами медичних зображень.

Отже, враховуючи результати проведеного дослідження, можна зробити висновок, що впровадження розробленої системи в медичних установах дозволяє автоматизувати процес збору та систематизації лабораторних аналізів, підвищуючи ефективність роботи медичного персоналу, що в свою чергу пришвидшує процес зберігання медичної інформації до бази даних, далі цю інформацію можна

використовувати для аналізу отриманих даних та розвитку досліджень у галузі охорони здоров'я.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. How OCR Works: An In-Depth Explanation of Optical Character Recognition. *We Label Data*. URL: https://www.welabeldata.com/post/how-ocr-works-an-in-depth-explanation-of-optical-character-recognition (дата звернення: 30.05.2024).

2. Tripathi P. 10 text recognition algorithms. *Docsumo - Document AI Platform Built for Scale & Efficiency*. URL: https://www.docsumo.com/blog/text-recognition-algorithms (дата звернення: 01.06.2024).

3. Advantages and disadvantages of the main programming languages - ITC Web Solutions. *ITC Web Solutions*. URL: https://itcwebsolutions.com/en/web-development-and-support/programming-and-tools/programming-languages/advantages-and-disadvantages-of-the-main-programming-languages/ (дата звернення: 18.05.2024).

4. Arsenault C. The Pros and Cons of 8 Popular Databases - KeyCDN. *KeyCDN*. URL: https://www.keycdn.com/blog/popular-databases (дата звернення: 18.05.2024).

5. Khalid. What Is OCR And What Is It Used For?. *Docparser*. URL: https://docparser.com/blog/what-is-ocr/ (дата звернення: 30.05.2024).

6. Stankevičiūtė G. What is Optical Character Recognition Technology? - iDenfy. *iDenfy*. URL: https://www.idenfy.com/blog/what-is-optical-character-recognition/ (дата звернення: 30.05.2024).

7. What Is Optical Character Recognition (OCR)? - IBM Blog. *IBM Blog*. URL: https://www.ibm.com/blog/optical-character-recognition/ (дата звернення: 30.05.2024).

8. DrMax. How Does Optical Character Recognition Work | Baeldung on Computer Science. *Baeldung on Computer Science*. URL: https://www.baeldung.com/cs/ocr#1-useful-text-from-an-image (дата звернення: 30.05.2024).

9. Manok A. All You Need to Know about Machine Learning OCR | Affinda. *Affinda | Become An AI-First Company*. URL: https://www.affinda.com/tech-ai/machine-learning-ocr (дата звернення: 30.05.2024).

10. Free PDF Reader & Viewer | FineReader PDF. *FineReader PDF*. URL: https://pdf.abbyy.com/finereader-pdf-viewer/ (дата звернення: 18.05.2024).

11. Tesseract.js | Pure Javascript OCR for 100 Languages!. *Tesseract.js | Pure Javascript OCR for 100 Languages!*. URL: https://tesseract.projectnaptha.com/ (дата звернення: 18.05.2024).

12. Vision AI: Image & Visual AI Tools. *Google Cloud*. URL: https://cloud.google.com/vision?hl=ru (дата звернення: 18.05.2024).

13. OCR Software, Data Extraction Tool - Amazon Textract - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/textract/?nc1=h_ls (дата звернення: 18.05.2024).

14. How to use OCR software for PDFs in 4 easy steps | Adobe Acrobat. *Adobe Acrobat.* URL: https://www.adobe.com/acrobat/how-to/ocr-software-convert-pdf-to-text.html (дата звернення: 18.05.2024).

15. Beznos M. Microservices vs monolithic architecture: How each of them can impact your business?. *Software Development Company - N-iX*. URL: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/ (дата звернення: 30.05.2024).

16. Hoang C. Monolith Architecture. *Medium*. URL: https://tech.tamara.co/monolith-architecture-5f00270f384e (дата звернення: 30.05.2024).

17. Livens J. Microservices vs. monolithic architecture: Understanding the difference. *Dynatrace news*. URL: https://www.dynatrace.com/news/blog/microservices-vs-monolithic-architecture/ (дата звернення: 30.05.2024).

18. Precious L. Building and structuring a Node.js MVC application - LogRocket Blog. *LogRocket Blog*. URL: https://blog.logrocket.com/building-structuring-node-js-mvc-application/ (дата звернення: 18.05.2024).

19. Balaji D. Handling Multiple File Uploads with Multer in Node.js. *Medium*. URL: https://dvmhn07.medium.com/handling-multiple-file-uploads-with-multer-in-node-js-c1901a51f8b8 (дата звернення: 18.05.2024).

20. GitHub - tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository). *GitHub*. URL: https://github.com/tesseract-ocr/tesseract (дата звернення: 18.05.2024).

21. Tiwari N. [ML Story] Computer Vision made easy with Google Cloud Vision API. *Medium*. URL: https://dvmhn07.medium.com/handling-multiple-file-uploads-with-multer-in-node-js-c1901a51f8b8 (дата звернення: 18.05.2024).

22. Regular Expressions :: Eloquent JavaScript. *Eloquent JavaScript*. URL: https://eloquentjavascript.net/09_regexp.html (дата звернення: 18.05.2024).

23. PostgreSQL: About. *PostgreSQL: The world's most advanced open source database*. URL: https://www.postgresql.org/about/ (дата звернення: 18.05.2024).

24. Grigutytė M. What is Bcrypt and how it works?. *NordVPN*. URL: https://nordvpn.com/blog/what-is-bcrypt/ (дата звернення: 18.05.2024).

25. Node.js – Run JavaScript Everywhere. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/en (дата звернення: 18.05.2024).

# ДОДАТОК А

# ТЕКСТ ПРОГРАМНОГО МОДУЛЯ

**«Формування бази даних лабораторних аналізів шляхом розпізнання тексту на зображенні»**

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ІАТЕ_ЦТЕ_ТР-03233_24Б

Аркушів 14

Київ-2024

Програмні засоби:

- Мова програмування – JavaScript;
- Бібліотека розпізнавання тексту – @google-cloud/vision;
- Бібліотека хешування паролів – bcrypt;
- Фреймворк – express;
- Бібліотека для генерації JWT токенів – jsonwebtoken;
- Бібліотека для створення сховища для зображень – multer;
- Бібліотека для доступу до бази даних – pg;
- Бібліотека розпізнавання тексту – tesseract.js.

**Файл package.json з описом залежностей**

```
{
  "name": "project",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "nodemon server.js"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "@google-cloud/vision": "^4.2.1",
    "bcrypt": "^5.1.1",
    "body-parser": "^1.20.2",
    "cookie-parser": "^1.4.6",
    "ejs": "^3.1.10",
    "express": "^4.19.2",
    "jsonwebtoken": "^9.0.2",
    "multer": "^1.4.5-lts.1",
    "pg": "^8.11.5",
```

```
    "tesseract.js": "^5.0.5"
  },
  "devDependencies": {
    "nodemon": "^3.1.0"
  }
}
```

**Файл db.js з описом підключень до бази даних**

```
const { Pool } = require('pg');

const pool = new Pool({
user: 'postgres',
password: '12345',
host: 'localhost',
port: 5432,
database: 'laboratoryDB',
});

module.exports = {
    pool
}
```

**Файл storage.js з описом файлового сховища**

```
const multer = require('multer')

const storage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, "./public/uploads")
    },
    filename: (req, file, cb) => {
```

```
        cb(null, file.originalname)
    }
})

exports.upload = multer({ storage: storage }).single("input_img")
```

**Файл session.js що відповідає за авторизацію користувачів**

```
const jwt = require('jsonwebtoken');
const {secretKey} = require('./secretKey')

function authenticateToken(req, res, next) {
    const token = req.cookies.access_token;

    if (!token) {
        return res.redirect('/');
        //return res.status(401).json({ error: 'Authorization token missing' });
    }

    jwt.verify(token, secretKey, (err, decoded) => {
        if (err) {
            res.clearCookie('access_token');
            return res.redirect('/');
            //return res.status(403).json({ error: 'Invalid token' });
        }

        req.user = decoded;
        next();
    });
}

function isUserLoggedIn(req, res, next) {
```

```
    const token = req.cookies.access_token;

    if (!token) {
        return next();
    }

    jwt.verify(token, secretKey, (err, decoded) => {
        if (err) {
            res.clearCookie('authToken');
            return next();
        }
        req.user = decoded;
        next();
    });
}

module.exports = {
    authenticateToken,
    isUserLoggedIn
}
```

**Файл analysis_controller.js, що відповідає за запити до бази даних**

```
const db = require('../services/db');

exports.getAllAnalysis = async (req, res) => {
    try {
        const result = await db.pool.query('select * from analysis order by id;')
        return res.send(result.rows)
    } catch (err) {
        console.error(err);
        return res.status(500).json({ message: 'Internal server error' });
```

```
    }
}

exports.createAnalysis = async (req, res) => {
    const userId = req.user.userId;
    const { analysis_code, analysis_person, analysis_date, analysis_name, analysis_result} = req.body;
    try {
        const result = await db.pool.query({
            text: 'insert into analysis (analysis_code, analysis_person, analysis_date, analysis_name, analysis_result, user_id) values ($1, $2, TO_DATE($3, \'DD.MM.YYYY\'), $4, $5, $6) returning *',
            values: [
                analysis_code,
                analysis_person,
                analysis_date,
                analysis_name,
                analysis_result,
                userId
            ]
        })
        return res.status(201).json({ message: `аналіз завантажено під id: ${result.rows[0].id}` });
    } catch (err) {
        console.error(err);
        return res.status(500).json({ error: 'Internal server error' });
    }
}

exports.updateAnalysis = async (req, res) => {
    try {
```

```
        const result = await db.pool.query({
            text: `update analysis set analysis_code = $1, analysis_person = $2, analysis_date = $3, analysis_name = $4, analysis_result = $5 WHERE id = $6 returning *`,
            values: [
                req.body.analysis_code,
                req.body.analysis_person ? req.body.analysis_person : null,
                req.body.analysis_date,
                req.body.analysis_name,
                req.body.analysis_result,
                req.body.id
            ]
        })

        if (result.rowCount == 0) {
            return res.status(404).json({ error: 'Аналіз не знайдено' })
        }

        return res.status(200).json({ message: `Аналіз ${req.body.id} оновлено` })
    } catch (err) {
        console.error(err);
        return res.status(500).json({ error: 'Internal server error' });
    }
}

exports.deleteAnalysis = async (req, res) => {
    const { id } = req.body;
    try {
        const result = await db.pool.query({
            text: 'delete from analysis where id = $1',
            values: [id]
```

```
            })

            if (result.rowCount == 0) {
                return res.status(404).json({ error: 'Аналіз не знайдено' })
            }

            return res.status(200).json({ message: `Аналіз ${req.body.id} видалено` })
        } catch (err) {
            console.error(err);
            return res.status(500).json({ error: 'Internal server error' });
        }
    }

    exports.analysisList = async (req, res) => {
        const userId = req.user.userId;
        const userRole = req.user.userRole;
        try {
            const page = parseInt(req.query.page) || 1;
            const limit = parseInt(req.query.limit) || 10;
            const offset = (page - 1) * limit;
            const column = req.query.column;

            var text, values;

            if(userRole === 'admin') {
                text = 'select id, analysis_code, analysis_person, analysis_date,
analysis_name, analysis_result from analysis order by ' + column + ' limit $1 offset $2';
                values = [limit, offset];
            }
            else {
```

```
            text = 'select id, analysis_code, analysis_person, analysis_date, analysis_name, analysis_result from analysis where user_id = $1 order by ' + column + ' limit $2 offset $3';
            values = [userId, limit, offset];
        }
        const result = await db.pool.query({
            text: text,
            values: values
        })

        return res.status(200).json(result.rows);
    } catch (err) {
        console.error(err);
        return res.status(500).json({ error: 'Internal server error' });
    }
}
```

**Файл gvocr_controller.js, відповідає за розпізнання тексту бібліотекою Google Cloud Vision**

```
const mystorage = require('../services/mystorage');
const vision = require('@google-cloud/vision');

const CREDENTIALS = JSON.parse(JSON.stringify({
    Тут зміст файлу json з персональною інформацією підключення
}));

const CONFIG = {
    credentials: {
        private_key: CREDENTIALS.private_key,
        client_email: CREDENTIALS.client_email
    }
```

```
};

const client = new vision.ImageAnnotatorClient(CONFIG);

exports.gvRecognize = (req, res) => {
   try {
      mystorage.upload(req, res, err => {
         const detectText = async (file_path) => {
            let [result] = await client.textDetection(file_path);
            return res.send([result.fullTextAnnotation.text])
         };
         detectText(`./public/uploads/${req.file.originalname}`);
      })
   } catch (err) {
      console.error(err);
      return res.status(500).json({ error: 'Internal server error' });
   }
}
```

**Файл tesocr_controller.js, відповідає за розпізнання тексту бібліотекою Tesseract**

```
const fs = require("fs")
const mystorage = require('../services/mystorage');
const Tesseract = require('tesseract.js');

exports.tesRecognize = (req, res) => {
   try {
      mystorage.upload(req, res, err => {
         fs.readFile(`./public/uploads/${req.file.originalname}`, (err, data) => {
            Tesseract.recognize(
               data,
```

```
                'ukr',
                { logger: m => console.log(m) }
            ).then(({ data: { text } }) => {
                return res.send([text]);
            })
        })
    })
  } catch (err) {
    console.error(err);
    return res.status(500).json({ error: 'Internal server error' });
  }
}
```

**Файл server.js, головний файл серверу, що потрібно запускати**

```
const express = require('express')
const cookieParser = require('cookie-parser');
const authFunctions = require('./services/session');
const app = express()

app.set('view engine', 'ejs')

app.use(express.static('public'))
//app.use(express.urlencoded({ extended: false }))
app.use(express.urlencoded({ extended: true }))
app.use(express.json())
app.use(cookieParser());

app.use(require('./routes/analysis_route'))
app.use(require('./routes/tesocr_route'))
app.use(require('./routes/gvocr_route'))
app.use(require('./routes/user_route'))
```

```
app.get('/', (req, res) => {
   res.render('index')
})

app.get('/ocr', authFunctions.authenticateToken, (req, res) => {
   res.render('ocr')
})

app.get('/analysis', authFunctions.authenticateToken, (req, res) => {
   res.render('analysis')
})

app.get('/login_page', (req, res) => {
   res.render('login_page')
})

app.get('/user_data', authFunctions.authenticateToken, (req, res) => {
   const userId = req.user.userId;
   const userRole = req.user.userRole;
   res.json({ userId: userId, userRole: userRole });
})

app.get('/is_logged', authFunctions.isUserLoggedIn, (req, res) => {
   if (req.user) {
     res.json({ logged: true });
   } else {
     res.json({ logged: false });
   }
})
```

```
app.get('/logout', (req, res) => {
   res.clearCookie('access_token');
   res.render('index');
});

const PORT = 3000

app.listen(PORT, () => {
   console.log(`Server started: http://localhost:${PORT}`)
})
```

**Файл ocr.js, функція структурування даних**

```
function useText(text) {
   const regex = /Дослідження:[ \n]*(.*?)[ \n]*ПІБ:[ \n]*(.*?)[ \n]*Дата:[ \n]*(.*?)[ \n]*Назва дослідження:[ \n]*(.*?)[ \n]*Результат:[ \n]*(.*)/s;
   const match = text.match(regex);

   if (match) {
      const [_, researchNumber, fullName, date, researchName, result] = match;

      var newDate = date.replace(/,/g, ".").replace(/\s+/g, "");
      var dateParts = newDate.split('.');
      var day = parseInt(dateParts[0]);
      var month = parseInt(dateParts[1]) - 1;
      var year = parseInt(dateParts[2]);
      var dateCheck = new Date(year, month, day);
      if (
         dateCheck.getFullYear() === year &&
         dateCheck.getMonth() === month &&
         dateCheck.getDate() === day &&
         year <= 2030 &&
```

```
                year >= 1990
            ) {
                const formattedDate = `${year}-${String(month + 1).padStart(2, '0')}-
${String(day).padStart(2, '0')}`;
                analysis_date.value = formattedDate;
            } else {
                console.log("Не вдалося розібрати строку");
                contentArea.innerHTML = text;
                result_div.style.display= 'flex';
                template_err.showModal();
                return;
            }

            result_div.style.display = 'none';
            contentArea.innerHTML = '';
            dialog_create_analysis.showModal();

            analysis_code.value = researchNumber;
            analysis_person.value = fullName;
            analysis_name.value = researchName;
            analysis_result.value = result;
        } else {
            console.log("Не вдалося розібрати строку");
            contentArea.innerHTML = text;
            result_div.style.display= 'flex';
            template_err.showModal();
        }
    }
```