ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки

Кваліфікаційна робота
ОКР «Бакалавр»

Комп'ютерна система класифікації об'єктів на зображеннях з використанням штучних нейронних мереж

Виконав:
ст. гр. КІУКІ-21-1
Стукота О.В.

Керівник:
ас. Климова І.М.

Мета та завдання кваліфікаційної роботи

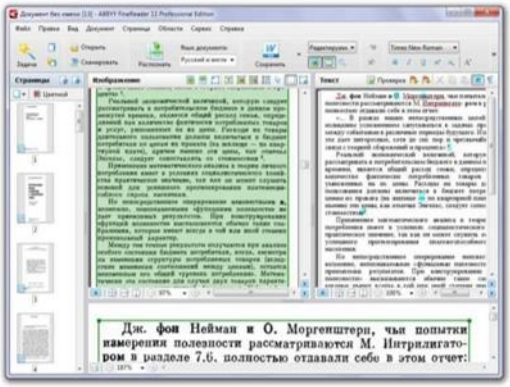
Метою кваліфікаційної роботи є розробка комп’ютерної системи класифікації об’єктів на зображеннях з використанням Python.

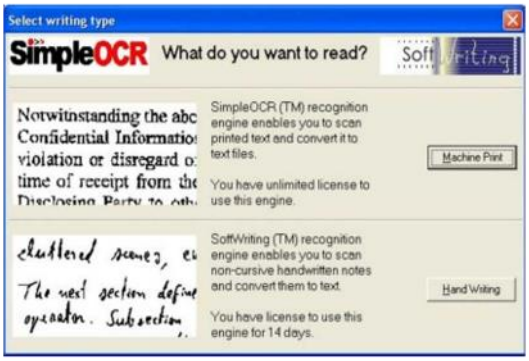
Завдання:

- ❖аналіз існуючих програмних засобів;
- ❖аналіз алгоритмів розпізнавання текстів на зображеннях;
- ❖розробка програмних засобів розпізнавання тексту на документах;
- ❖тестування розроблених засобів.

Аналіз існуючих рішень

3





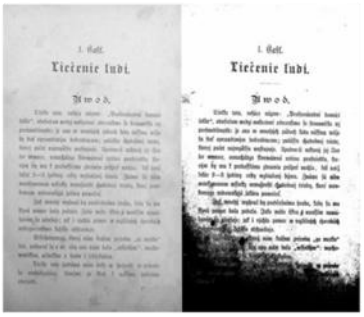
Аналіз існуючих рішень

4

Назва	Витрати	Платформа	Точність	Документація
ABBYY Finereader	Відсутні	Довільна	98%	Задовільно
Simple OCR	Відсутні	Довільна	50-90%	Погано
Free OCR	Відсутні	Довільна	60-90%	Добре
OCR Finder	Відсутні	Linux	60-90%	Задовільно
Tesseradct OCR	Відсутні	Хмара	60-99%	Задовільно
Google Vision	$1,5/1000	Хмара	>90%	Добре

Tesseract ORC та Google Vision

5



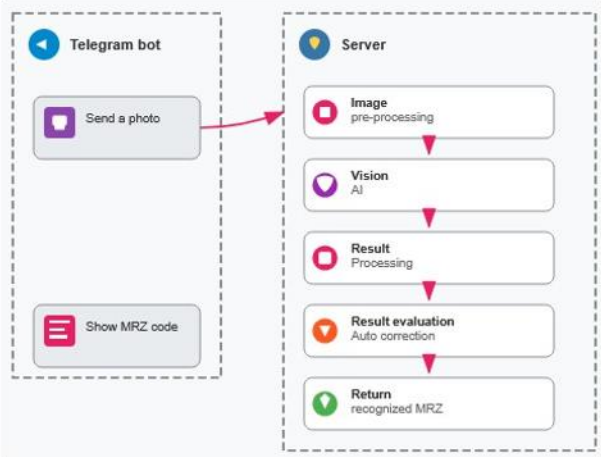
Обгрунтування вибору Python

6

Для реалізації системи була обрана мова програмування Python. Ця мова є сучасною та динамічно розвивається, має велику спільноту розробників, що дозволяє легко знаходити відповіді на всі питання, що виникають під час розробки. Мова має велику кількість бібліотек, що підтримуються та реалізують різноманітний функціонал. Це дає змогу максимально ефективно використовувати сучасні методи розробки програмного забезпечення та будувати системи, що відповідають усім сучасним стандартам та методам розробки, а також дозволяють легко підтримувати таке програмне забезпечення в майбутньому

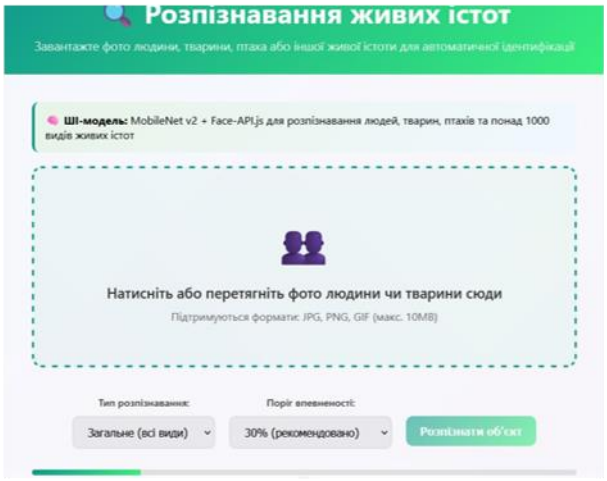
Етапи виконання програми

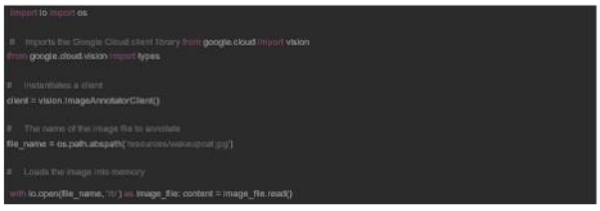
7



Інтерфейс. Програмний код

8





```
import io import os

#  Imports the Google Cloud client library from google.cloud import vision
from google.cloud.vision import types

#  Instantiates a client
client = vision.ImageAnnotatorClient()

#  The name of the image file to annotate
file_name = os.path.abspath('resources/wakeupcat.jpg')

#  Loads the image into memory
with io.open(file_name, 'rb') as image_file: content = image_file.read()
```

```
        with open(img_path, 'rb') as image_file: content = image_file.read()

        content = img
image = vision.types.Image(                                    =content)


response = client.document_text_detection( =image) return response
```

Створення програмного інтерфейсу та результати роботи

9

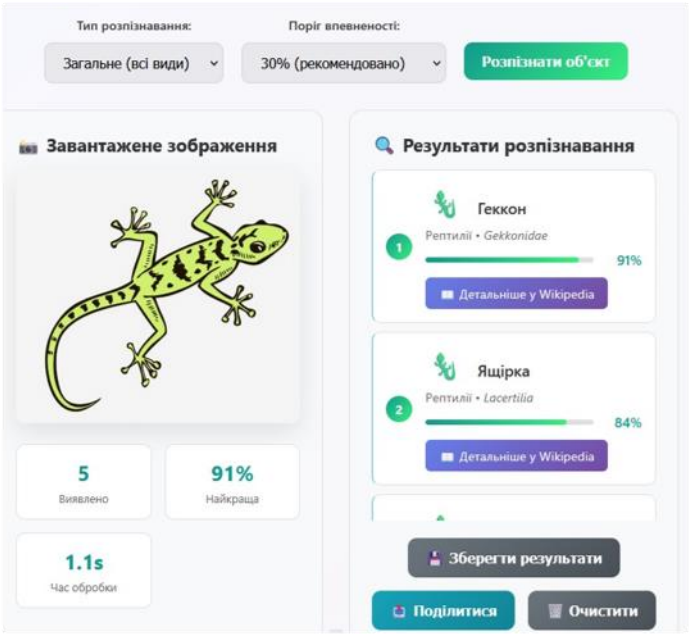
```
from flask import Flask, jsonify, send_file
from flask import request
import base64
import json
import os
import shutil
import time
import recognizer
import result_processing
from settings import host, database, user, password
import rotate_img


app = Flask(__name__)
DEBUG = False

@app.route('/api', methods=['POST'])
def get():...

if __name__ == '__main__':

    if DEBUG:
        try:
            app.run()
        except Exception as e:
            print('No text error')
    else:
        # Bind to PORT if defined, otherwise default to 5000.
        port = int(os.environ.get('PORT', 5000))
        app.run(host='0.0.0.0', port=port, debug=False)
```



10

Висновки

В ході реалізації даної кваліфікаційної роботи були розроблена комп'ютерна система класифікації даних з використанням мови Python. Проведено аналіз існуючих програмних засобів, аналіз алгоритмів розпізнавання текстів на зображеннях. Розроблені схема роботи програмних засобів розпізнавання. Проведено тестування розроблених програмних засобів.

ДОДАТОК Б
ПРОГРАМНИЙ КОД

```
# app.py - Застосунок розпізнавання живих істот
from flask import Flask, render_template, request, jsonify,
send_from_directory
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
import numpy as np
import cv2
import os
import time
import requests
import json
from PIL import Image
import io
import base64
from datetime import datetime
import sqlite3
import uuid
import logging

# Налаштування логування
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

app = Flask(__name__)
app.config['MAX_CONTENT_LENGTH'] = 10 * 1024 * 1024  # 10MB
максимальний розмір файлу
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['SECRET_KEY'] = 'your-secret-key-here'

# Створення необхідних папок
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)
os.makedirs('templates', exist_ok=True)
os.makedirs('static/css', exist_ok=True)
os.makedirs('static/js', exist_ok=True)

class AnimalRecognizer:
    """Клас для розпізнавання живих істот"""

    def __init__(self):
        """Ініціалізація моделі розпізнавання"""
        logger.info("Завантаження моделі MobileNetV2...")
        try:
            self.model = MobileNetV2(weights='imagenet',
```

```
include_top=True)
            self.model.compile(
                optimizer='adam',
                loss='categorical_crossentropy',
                metrics=['accuracy']
            )
            logger.info("Модель успішно завантажена!")
        except Exception as e:
            logger.error(f"Помилка завантаження моделі: {e}")
            raise

        # Словник перекладів українських назв
        self.ukrainian_names = {
            'gecko': 'Гекон',
            'lizard': 'Ящірка',
            'iguana': 'Ігуана',
            'chameleon': 'Хамелеон',
            'snake': 'Змія',
            'turtle': 'Черепаха',
            'frog': 'Жаба',
            'toad': 'Ропуха',
            'salamander': 'Саламандра',
            'bird': 'Птах',
            'eagle': 'Орел',
            'hawk': 'Яструб',
            'owl': 'Сова',
            'parrot': 'Папуга',
            'cat': 'Кіт',
            'dog': 'Собака',
            'horse': 'Кінь',
            'elephant': 'Слон',
            'tiger': 'Тигр',
            'lion': 'Лев',
            'bear': 'Ведмідь',
            'wolf': 'Вовк',
            'fox': 'Лисиця',
            'rabbit': 'Кролик',
            'deer': 'Олень',
            'cow': 'Корова',
            'sheep': 'Вівця',
            'pig': 'Свиня',
            'fish': 'Риба',
            'shark': 'Акула',
            'whale': 'Кит',
            'dolphin': 'Дельфін',
            'spider': 'Павук',
            'butterfly': 'Метелик',
            'bee': 'Бджола',
            'ant': 'Мураха'
        }

        # Таксономічні родини
        self.taxonomic_families = {
```

```
            'gecko': 'Gekkonidae',
            'lizard': 'Lacertilia',
            'iguana': 'Iguanidae',
            'chameleon': 'Chamaeleonidae',
            'snake': 'Serpentes',
            'turtle': 'Testudines',
            'frog': 'Anura',
            'salamander': 'Caudata',
            'bird': 'Aves',
            'eagle': 'Accipitridae',
            'hawk': 'Accipitridae',
            'owl': 'Strigidae',
            'parrot': 'Psittacidae',
            'cat': 'Felidae',
            'dog': 'Canidae',
            'horse': 'Equidae',
            'elephant': 'Elephantidae',
            'tiger': 'Felidae',
            'lion': 'Felidae',
            'bear': 'Ursidae',
            'wolf': 'Canidae',
            'fox': 'Canidae',
            'rabbit': 'Leporidae',
            'deer': 'Cervidae',
            'cow': 'Bovidae',
            'sheep': 'Bovidae',
            'pig': 'Suidae',
            'fish': 'Pisces',
            'shark': 'Selachimorpha',
            'whale': 'Cetacea',
            'dolphin': 'Delphinidae',
            'spider': 'Araneae',
            'butterfly': 'Lepidoptera',
            'bee': 'Apidae',
            'ant': 'Formicidae'
        }

    def preprocess_image(self, img_path):
        """Попередня обробка зображення для нейронної мережі"""
        try:
            # Завантаження зображення з потрібним розміром
            img = image.load_img(img_path, target_size=(224,
224))

            # Перетворення в numpy array
            img_array = image.img_to_array(img)

            # Додавання batch розміру
            img_array = np.expand_dims(img_array, axis=0)

            # Попередня обробка для MobileNetV2
            img_array = preprocess_input(img_array)
```

```
            return img_array
        except Exception as e:
            logger.error(f"Помилка обробки зображення
{img_path}: {e}")
            return None

    def detect_objects_in_image(self, img_path):
        """Виявлення об'єктів на зображенні за допомогою
OpenCV"""
        try:
            # Завантаження зображення
            image_cv = cv2.imread(img_path)
            if image_cv is None:
                return 1

            # Перетворення в HSV для кращого виявлення об'єктів
            hsv = cv2.cvtColor(image_cv, cv2.COLOR_BGR2HSV)

            # Створення маски для виявлення об'єктів
            # (це спрощений алгоритм, можна замінити на YOLO або
інший детектор)
            mask = cv2.inRange(hsv, (0, 50, 50), (180, 255,
255))

            # Знаходження контурів
            contours, _ = cv2.findContours(mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

            # Фільтрація контурів за розміром
            significant_contours = [c for c in contours if
cv2.contourArea(c) > 1000]

            return max(1, len(significant_contours))

        except Exception as e:
            logger.error(f"Помилка виявлення об'єктів: {e}")
            return 1

    def recognize_animal(self, img_path,
confidence_threshold=0.3, recognition_type='general'):
        """Основна функція розпізнавання тварини на
зображенні"""
        start_time = time.time()

        # Попередня обробка зображення
        processed_img = self.preprocess_image(img_path)
        if processed_img is None:
            return None

        # Виявлення кількості об'єктів
        objects_detected =
self.detect_objects_in_image(img_path)
```

```
        # Передбачення за допомогою моделі
        try:
            predictions = self.model.predict(processed_img,
verbose=0)
            decoded_predictions =
decode_predictions(predictions, top=10)[0]
        except Exception as e:
            logger.error(f"Помилка передбачення: {e}")
            return None

        processing_time = round(time.time() - start_time, 1)

        # Фільтрація та обробка результатів
        results = []
        max_confidence = 0
        valid_predictions = 0

        for i, (imagenet_id, label, confidence) in
enumerate(decoded_predictions):
            if confidence >= confidence_threshold:
                # Перевірка типу розпізнавання
                if self._is_valid_recognition_type(label,
recognition_type):
                    valid_predictions += 1
                    max_confidence = max(max_confidence,
confidence)

                    # Переклад назви на українську
                    ukrainian_name =
self.get_ukrainian_name(label)
                    family = self.get_taxonomic_family(label)
                    category = self.get_category(label)

                    results.append({
                        'rank': i + 1,
                        'name': ukrainian_name,
                        'scientific_name': label.replace('_', '
').title(),
                        'family': family,
                        'confidence': round(confidence * 100,
0),
                        'category': category,
                        'imagenet_id': imagenet_id
                    })

        # Сортування за впевненістю
        results.sort(key=lambda x: x['confidence'],
reverse=True)

        # Повернення результатів
        return {
            'results': results[:5],  # Топ 5 результатів
            'objects_detected': objects_detected,
```

```
            'max_confidence': round(max_confidence * 100, 0) if
max_confidence > 0 else 0,
            'processing_time': processing_time,
            'total_predictions': len(decoded_predictions),
            'valid_predictions': valid_predictions
        }

    def _is_valid_recognition_type(self, label,
recognition_type):
        """Перевірка чи відповідає результат типу
розпізнавання"""
        if recognition_type == 'general':
            return True

        label_lower = label.lower()

        if recognition_type == 'animals':
            animal_keywords = ['cat', 'dog', 'horse',
'elephant', 'tiger', 'lion', 'bear', 'wolf', 'fox', 'rabbit',
'deer', 'cow', 'sheep', 'pig']
            return any(keyword in label_lower for keyword in
animal_keywords)

        elif recognition_type == 'birds':
            bird_keywords = ['bird', 'eagle', 'hawk', 'owl',
'parrot', 'duck', 'swan', 'crow', 'robin', 'sparrow']
            return any(keyword in label_lower for keyword in
bird_keywords)

        elif recognition_type == 'reptiles':
            reptile_keywords = ['gecko', 'lizard', 'iguana',
'chameleon', 'snake', 'turtle', 'crocodile', 'alligator']
            return any(keyword in label_lower for keyword in
reptile_keywords)

        return True

    def get_ukrainian_name(self, english_name):
        """Отримання української назви тварини"""
        english_name_lower = english_name.lower()

        # Пошук точного співпадіння
        for key, value in self.ukrainian_names.items():
            if key in english_name_lower:
                return value

        # Якщо не знайдено, повертаємо очищену англійську назву
        return english_name.replace('_', ' ').replace('-', '
').title()

    def get_taxonomic_family(self, english_name):
        """Отримання таксономічної родини"""
        english_name_lower = english_name.lower()
```

```
        for key, value in self.taxonomic_families.items():
            if key in english_name_lower:
                return value

        return "Невідомо"

    def get_category(self, english_name):
        """Визначення категорії тварини"""
        english_name_lower = english_name.lower()

        # Рептилії
        if any(word in english_name_lower for word in ['gecko',
'lizard', 'snake', 'turtle', 'iguana', 'chameleon']):
            return 'Рептилії'

        # Птахи
        elif any(word in english_name_lower for word in ['bird',
'eagle', 'hawk', 'owl', 'parrot', 'duck', 'swan']):
            return 'Птахи'

        # Ссавці
        elif any(word in english_name_lower for word in ['cat',
'dog', 'horse', 'elephant', 'tiger', 'lion', 'bear']):
            return 'Ссавці'

        # Амфібії
        elif any(word in english_name_lower for word in ['frog',
'toad', 'salamander']):
            return 'Амфібії'

        # Риби
        elif any(word in english_name_lower for word in ['fish',
'shark', 'whale', 'dolphin']):
            return 'Риби та морські тварини'

        # Комахи
        elif any(word in english_name_lower for word in
['spider', 'butterfly', 'bee', 'ant', 'beetle']):
            return 'Комахи та павукоподібні'

        else:
            return 'Тварини'

class DatabaseManager:
    """Клас для управління базою даних"""

    def __init__(self, db_path='recognition_history.db'):
        """Ініціалізація менеджера бази даних"""
        self.db_path = db_path
        self.init_db()

    def init_db(self):
```

```
        """Створення таблиць бази даних"""
        try:
            conn = sqlite3.connect(self.db_path)
            cursor = conn.cursor()

            cursor.execute('''
                CREATE TABLE IF NOT EXISTS recognition_history (
                    id TEXT PRIMARY KEY,
                    timestamp DATETIME DEFAULT
CURRENT_TIMESTAMP,
                    image_path TEXT NOT NULL,
                    original_filename TEXT,
                    results TEXT NOT NULL,
                    processing_time REAL,
                    user_ip TEXT,
                    recognition_type TEXT,
                    confidence_threshold REAL,
                    objects_detected INTEGER,
                    max_confidence REAL
                )
            ''')

            cursor.execute('''
                CREATE TABLE IF NOT EXISTS user_sessions (
                    session_id TEXT PRIMARY KEY,
                    user_ip TEXT,
                    first_visit DATETIME DEFAULT
CURRENT_TIMESTAMP,
                    last_visit DATETIME DEFAULT
CURRENT_TIMESTAMP,
                    total_recognitions INTEGER DEFAULT 0
                )
            ''')

            conn.commit()
            conn.close()
            logger.info("База даних ініціалізована успішно")

        except Exception as e:
            logger.error(f"Помилка ініціалізації бази даних:
{e}")

    def save_recognition(self, image_path, original_filename,
results, processing_time,
                         user_ip, recognition_type,
confidence_threshold):
        """Збереження результату розпізнавання в базу даних"""
        try:
            conn = sqlite3.connect(self.db_path)
            cursor = conn.cursor()

            recognition_id = str(uuid.uuid4())
            timestamp = datetime.now()
```

```
            cursor.execute('''
                INSERT INTO recognition_history
                (id, timestamp, image_path, original_filename,
results, processing_time,
                 user_ip, recognition_type,
confidence_threshold, objects_detected, max_confidence)
                VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            ''', (
                recognition_id, timestamp, image_path,
original_filename,
                json.dumps(results, ensure_ascii=False),
processing_time,
                user_ip, recognition_type, confidence_threshold,
                results.get('objects_detected', 0),
results.get('max_confidence', 0)
            ))

            conn.commit()
            conn.close()

            logger.info(f"Результат розпізнавання збережено:
{recognition_id}")
            return recognition_id

        except Exception as e:
            logger.error(f"Помилка збереження в базу даних:
{e}")
            return None

    def get_history(self, limit=50):
        """Отримання історії розпізнавань"""
        try:
            conn = sqlite3.connect(self.db_path)
            cursor = conn.cursor()

            cursor.execute('''
                SELECT id, timestamp, original_filename,
results, processing_time,
                       recognition_type, confidence_threshold,
objects_detected, max_confidence
                FROM recognition_history
                ORDER BY timestamp DESC
                LIMIT ?
            ''', (limit,))

            history = []
            for row in cursor.fetchall():
                try:
                    results = json.loads(row[3])
                    history.append({
                        'id': row[0],
                        'timestamp': row[1],
```

```
                        'filename': row[2],
                        'results': results,
                        'processing_time': row[4],
                        'recognition_type': row[5],
                        'confidence_threshold': row[6],
                        'objects_detected': row[7],
                        'max_confidence': row[8]
                    })
                except json.JSONDecodeError:
                    continue

            conn.close()
            return history

        except Exception as e:
            logger.error(f"Помилка отримання історії: {e}")
            return []

    def get_statistics(self):
        """Отримання статистики використання"""
        try:
            conn = sqlite3.connect(self.db_path)
            cursor = conn.cursor()

            # Загальна кількість розпізнавань
            cursor.execute('SELECT COUNT(*) FROM
recognition_history')
            total_recognitions = cursor.fetchone()[0]

            # Середній час обробки
            cursor.execute('SELECT AVG(processing_time) FROM
recognition_history')
            avg_processing_time = cursor.fetchone()[0] or 0

            # Найпопулярніші тварини
            cursor.execute('''
                SELECT results, COUNT(*) as count
                FROM recognition_history
                GROUP BY results
                ORDER BY count DESC
                LIMIT 10
            ''')

            popular_animals = []
            for row in cursor.fetchall():
                try:
                    results = json.loads(row[0])
                    if results.get('results'):
                        top_result = results['results'][0]
                        popular_animals.append({
                            'name': top_result['name'],
                            'count': row[1]
                        })
```

```
                except:
                    continue

            conn.close()

            return {
                'total_recognitions': total_recognitions,
                'avg_processing_time':
round(avg_processing_time, 2),
                'popular_animals': popular_animals[:5]
            }

        except Exception as e:
            logger.error(f"Помилка отримання статистики: {e}")
            return {}

class WikipediaAPI:
    """Клас для роботи з Wikipedia API"""

    @staticmethod
    def get_animal_info(animal_name, lang='uk'):
        """Отримання інформації про тварину з Wikipedia"""
        try:
            # Спроба отримати українську версію
            search_url =
f"https://{lang}.wikipedia.org/api/rest_v1/page/summary/{animal_
name}"
            response = requests.get(search_url, timeout=5)

            if response.status_code == 200:
                data = response.json()
                return {
                    'title': data.get('title', ''),
                    'extract': data.get('extract', ''),
                    'url': data.get('content_urls',
{}).get('desktop', {}).get('page', ''),
                    'image': data.get('thumbnail',
{}).get('source', '') if data.get('thumbnail') else '',
                    'language': lang
                }
            elif lang == 'uk':
                # Якщо українська версія не знайдена, спробувати
англійську
                return WikipediaAPI.get_animal_info(animal_name,
'en')

        except Exception as e:
            logger.error(f"Помилка отримання інформації з
Wikipedia: {e}")

        return None

    @staticmethod
```

```
    def search_animals(query, lang='uk', limit=5):
        """Пошук тварин у Wikipedia"""
        try:
            search_url =
f"https://{lang}.wikipedia.org/api/rest_v1/page/search/{query}"
            params = {'limit': limit}
            response = requests.get(search_url, params=params,
timeout=5)

            if response.status_code == 200:
                data = response.json()
                return data.get('pages', [])

        except Exception as e:
            logger.error(f"Помилка пошуку в Wikipedia: {e}")

        return []

# Ініціалізація компонентів системи
logger.info("Ініціалізація компонентів системи...")
recognizer = AnimalRecognizer()
db_manager = DatabaseManager()

# Маршрути Flask застосунку
@app.route('/')
def index():
    """Головна сторінка застосунку"""
    return render_template('index.html')

@app.route('/upload', methods=['POST'])
def upload_file():
    """Завантаження та обробка файлу зображення"""
    try:
        # Перевірка наявності файлу
        if 'file' not in request.files:
            return jsonify({'error': 'Файл не знайдено в
запиті'}), 400

        file = request.files['file']
        if file.filename == '':
            return jsonify({'error': 'Файл не вибрано'}), 400

        # Перевірка типу файлу
        allowed_extensions = {'.jpg', '.jpeg', '.png', '.gif',
'.bmp', '.webp'}
        file_ext = os.path.splitext(file.filename)[1].lower()
        if file_ext not in allowed_extensions:
            return jsonify({
                'error': f'Непідтримуваний формат файлу.
Дозволені: {", ".join(allowed_extensions)}'
            }), 400

        # Перевірка розміру файлу
```

```
        if len(file.read()) > app.config['MAX_CONTENT_LENGTH']:
            return jsonify({'error': 'Розмір файлу перевищує
10МБ'}), 400
        file.seek(0)  # Повернути покажчик на початок файлу

        # Збереження файлу
        filename = f"{uuid.uuid4()}{file_ext}"
        file_path = os.path.join(app.config['UPLOAD_FOLDER'],
filename)
        file.save(file_path)

        # Отримання параметрів розпізнавання
        confidence_threshold =
float(request.form.get('confidence', 0.3))
        recognition_type = request.form.get('type', 'general')

        # Валідація параметрів
        if not (0.1 <= confidence_threshold <= 1.0):
            confidence_threshold = 0.3

        if recognition_type not in ['general', 'animals',
'birds', 'reptiles']:
            recognition_type = 'general'

        # Розпізнавання тварини
        logger.info(f"Початок розпізнавання файлу:
{file.filename}")
        results = recognizer.recognize_animal(
            file_path,
            confidence_threshold,
            recognition_type
        )

        if results is None:
            os.remove(file_path)  # Видалити файл при помилці
            return jsonify({'error': 'Помилка обробки
зображення'}), 500

        # Збереження результатів в базу даних
        user_ip = request.environ.get('HTTP_X_FORWARDED_FOR',
request.remote_addr)
        recognition_id = db_manager.save_recognition(
            file_path, file.filename, results,
results['processing_time'],
            user_ip, recognition_type, confidence_threshold
        )

        # Кодування зображення для відображення
        try:
            with open(file_path, 'rb') as img_file:
                img_data =
base64.b64encode(img_file.read()).decode('utf-8')
                img_data_url =
```

```
f"{img_data}"
        except Exception as e:
            logger.error(f"Помилка кодування зображення: {e}")
            img_data_url = ""

        # Формування відповіді
        response_data = {
            'recognition_id': recognition_id,
            'image_data': img_data_url,
            'results': results,
            'filename': file.filename,
            'parameters': {
                'confidence_threshold': confidence_threshold,
                'recognition_type': recognition_type
            }
        }

        logger.info(f"Розпізнавання завершено успішно. ID:
{recognition_id}")
        return jsonify(response_data)

    except Exception as e:
        logger.error(f"Помилка при завантаженні файлу: {e}")
        return jsonify({'error': 'Внутрішня помилка сервера'}),
500

@app.route('/wikipedia/<animal_name>')
def get_wikipedia_info(animal_name):
    """Отримання інформації про тварину з Wikipedia"""
    try:
        info = WikipediaAPI.get_animal_info(animal_name)
        if info:
            return jsonify(info)
        else:
            return jsonify({'error': 'Інформацію не знайдено'}),
404
    except Exception as e:
        logger.error(f"Помилка Wikipedia API: {e}")
        return jsonify({'error': 'Помилка підключення до
Wikipedia'}), 500

@app.route('/history')
def get_history():
    """Отримання історії розпізнавань"""
    try:
        limit = int(request.args.get('limit', 50))
        history = db_manager.get_history(limit)
        return jsonify(history)
    except Exception as e:
        logger.error(f"Помилка отримання історії: {e}")
        return jsonify({'error': 'Помилка отримання історії'}),
500
```

```
@app.route('/statistics')
def get_statistics():
    """Отримання статистики використання"""
    try:
        stats = db_manager.get_statistics()
        return jsonify(stats)
    except Exception as e:
        logger.error(f"Помилка отримання статистики: {e}")
        return jsonify({'error': 'Помилка отримання
статистики'}), 500

@app.route('/search/<query>')
def search_animals(query):
    """Пошук тварин в Wikipedia"""
    try:
        results = WikipediaAPI.search_animals(query)
        return jsonify(results)
    except Exception as e:
        logger.error(f"Помилка пошуку: {e}")
        return jsonify({'error': 'Помилка пошуку'}), 500

@app.route('/health')
def health_check():
    """Перевірка стану сервісу"""
    return jsonify({
        'status': 'healthy',
        'timestamp': datetime.now().isoformat(),
        'model_loaded': recognizer.model is not None
    })

@app.errorhandler(404)
def not_found(error):
    """Обробка помилки 404"""
    return jsonify({'error': 'Сторінку не знайдено'}), 404

@app.errorhandler(500)
def internal_error(error):
    """Обробка внутрішніх помилок"""
    logger.error(f"Внутрішня помилка: {error}")
    return jsonify({'error': 'Внутрішня помилка сервера'}), 500

if __name__ == '__main__':
    logger.info("Запуск Flask сервера...")
    logger.info("Сервер буде доступний за адресою:
http://localhost:5000")

    # Запуск в режимі розробки
    app.run(
        debug=True,
        host='0.0.0.0',
        port=5000,
        threaded=True
    )oral;
```