ImageUploadHandler - Image Processing Utility

📋 Tổng quan

`ImageUploadHandler` là utility class xử lý upload và chuyển đổi hình ảnh, giải quyết vấn đề "cannot write mode RGBA as JPEG" và các vấn đề format khác.

🎯 Tính năng

☑ Automatic Format Conversion

- Tự động chuyển đổi RGBA → RGB (alpha channel → white background)
- Hỗ trợ: RGBA, LA, P (Palette), L (Grayscale), CMYK
- Đảm bảo tương thích với JPEG format

☑ Quality Metrics

- Blur score (độ mờ)
- Brightness (độ sáng)
- Contrast (độ tương phản)
- Overall quality score (0-100)

☑ Temporary File Management

- Tự động lưu vào temp file
- Auto-cleanup
- Hỗ trợ nhiều format output: JPEG, PNG, BMP, WEBP, TIFF

☑ OpenCV Integration

- Convert PIL → OpenCV array (BGR)
- Tích hợp với các xử lý computer vision

🚀 Cách sử dụng

1. Basic Usage

```
from service.utils.ImageUploadHandler import ImageUploadHandler

# Initialize handler
handler = ImageUploadHandler(auto_convert_to_rgb=True)

# Process uploaded bytes
image_bytes = await file.read()
result = handler.process_upload(
    image_bytes,
    save_temp=True,
```

```
    format='JPEG',
    calculate_metrics=True
)

# Access results
image = result['image']  # PIL Image
temp_path = result['temp_path']  # Path to temp file
info = result['info']  # Image info dict
metrics = result['metrics']  # Quality metrics

# Cleanup
handler.cleanup_temp(temp_path)
```

2. In FastAPI Endpoint

```
from fastapi import UploadFile, File
from service.utils.ImageUploadHandler import ImageUploadHandler

@router.post("/process")
async def process_image(file: UploadFile = File(...)):
    # Read uploaded file
    contents = await file.read()

    # Process with handler
    handler = ImageUploadHandler(auto_convert_to_rgb=True)
    result = handler.process_upload(contents, save_temp=True)

    try:
        # Use the processed image
        temp_path = result['temp_path']

        # Your processing logic here
        # ...

        return {
            "status": "success",
            "image_info": result['info'],
            "metrics": result['metrics']
        }
    finally:
        # Always cleanup
        handler.cleanup_temp(result['temp_path'])
```

3. Manual Conversion

```
from PIL import Image
from service.utils.ImageUploadHandler import ImageUploadHandler
```

```
# Load RGBA image
rgba_image = Image.open('image_with_alpha.png')

# Convert to RGB
handler = ImageUploadHandler()
rgb_image = handler.convert_to_rgb(rgba_image)

# Now can save as JPEG
rgb_image.save('output.jpg', 'JPEG')
```

4. Get OpenCV Array

```
# Convert PIL Image to OpenCV array
handler = ImageUploadHandler()
cv2_array = handler.to_cv2_array(pil_image)

# Use with OpenCV
import cv2
gray = cv2.cvtColor(cv2_array, cv2.COLOR_BGR2GRAY)
```

5. Calculate Metrics Only

```
handler = ImageUploadHandler()
metrics = handler.calculate_quality_metrics(image)

print(f"Blur score: {metrics['blur_score']}")
print(f"Quality score: {metrics['quality_score']}")
```

📊 Return Structure

Image Info

```
{
  "original_mode": "RGBA",
  "original_format": "PNG",
  "width": 1920,
  "height": 1080,
  "file_size": 1234567,
  "converted": true,
  "final_mode": "RGB"
}
```

Quality Metrics

```
{
  "blur_score": 2551.69,
  "brightness": 127.45,
  "contrast": 45.23,
  "quality_score": 89.5
}
```

Supported Conversions

From Mode	To Mode	Method
RGBA	RGB	Alpha → white background
LA	RGB	Grayscale alpha → white
P	RGB	Palette expansion
L	RGB	Grayscale → RGB
CMYK	RGB	Color space conversion

Configuration

```
# Custom settings
handler = ImageUploadHandler(
    auto_convert_to_rgb=True  # Auto convert non-RGB to RGB
)

# Custom file size limit
handler.MAX_FILE_SIZE = 20 * 1024 * 1024  # 20MB

# Custom output format
temp_path = handler.save_to_temp(
    image,
    format='PNG',
    quality=95,  # JPEG quality
    suffix='.png'
)
```

Error Handling

```
from service.utils.ImageUploadHandler import ImageUploadHandler

handler = ImageUploadHandler()

try:
    result = handler.process_upload(image_bytes)
```

```
except ValueError as e:
    # File too large or invalid format
    print(f"Error: {e}")
except Exception as e:
    # Other errors
    print(f"Unexpected error: {e}")
```

Best Practices

DO:

```
# Always cleanup temp files
try:
    result = handler.process_upload(bytes)
    # Use temp_path
finally:
    handler.cleanup_temp(result['temp_path'])

# Use context manager pattern
with tempfile.NamedTemporaryFile() as tmp:
    # Process
    pass
```

DON'T:

```
# Don't forget to cleanup
result = handler.process_upload(bytes)
# ... no cleanup = memory leak!

# Don't save RGBA as JPEG without conversion
rgba_img.save('output.jpg')  # ERROR!
```

Common Issues

Issue 1: "cannot write mode RGBA as JPEG"

Solution: Use `ImageUploadHandler` with `auto_convert_to_rgb=True`

```
handler = ImageUploadHandler(auto_convert_to_rgb=True)
result = handler.process_upload(bytes, format='JPEG')
```

Issue 2: High memory usage

Solution: Always cleanup temp files

```
try:
    result = handler.process_upload(bytes)
finally:
    handler.cleanup_temp(result['temp_path'])
```

Issue 3: File size too large

Solution: Adjust MAX_FILE_SIZE

```
handler.MAX_FILE_SIZE = 50 * 1024 * 1024  # 50MB
```

Integration Examples

With scan.py endpoint:

```
from service.utils.ImageUploadHandler import ImageUploadHandler

@router.post("/scan/")
async def scan_card(image_file: UploadFile = File(...)):
    contents = await file.read()

    # Process with handler
    handler = ImageUploadHandler(auto_convert_to_rgb=True)
    result = handler.process_upload(contents, save_temp=True)

    try:
        temp_path = result['temp_path']
        image_quality = result['info'] | result['metrics']

        # Your OCR/detection logic
        # ...

        return {"status": "success", "quality": image_quality}
    finally:
        handler.cleanup_temp(result['temp_path'])
```

API Reference

Class: `ImageUploadHandler`

Methods:

- `__init__(auto_convert_to_rgb: bool = True)`
  - Initialize handler

- `convert_to_rgb(image: Image.Image) -> Image.Image`
  - Static method to convert any image mode to RGB
- `load_from_bytes(image_bytes: bytes) -> Tuple[Image.Image, dict]`
  - Load image from bytes with info
- `save_to_temp(image: Image.Image, format: str = 'JPEG') -> str`
  - Save to temporary file
- `to_cv2_array(image: Image.Image) -> np.ndarray`
  - Convert PIL → OpenCV BGR array
- `calculate_quality_metrics(image: Union[Image.Image, np.ndarray]) -> dict`
  - Calculate blur, brightness, contrast, quality score
- `process_upload(image_bytes: bytes, ...) -> dict`
  - Complete processing pipeline
- `cleanup_temp(temp_path: str) -> bool`
  - Static method to cleanup temp file

✨ Benefits

1. **No more RGBA → JPEG errors** ☑
2. **Automatic format handling** ☑
3. **Built-in quality metrics** ☑
4. **Memory-efficient temp file management** ☑
5. **OpenCV integration** ☑
6. **Production-ready** ☑

🔗 Related Files

- Source: `service/utils/ImageUploadHandler.py`
- Usage: `src/api/scan.py`
- Tests: `tests/test_image_handler.py` (TODO)