Media API Implementation Summary

Overview

The Media Handling system has been successfully implemented for the Content Service, providing comprehensive file upload, download, processing, and management capabilities. This system supports images, videos, audio files, and documents with automatic processing, thumbnail generation, and secure storage.

Implementation Date: December 22, 2024
Service: Content Service
Location: `/home/ubuntu/content_service`

Architecture

System Components

```
Media Handling System
├── Models Layer (app/models/media.py)
│   └── Media model with comprehensive fields
├── Schemas Layer (app/schemas/media.py)
│   ├── MediaCreate, MediaUpdate, MediaResponse
│   └── MediaList (pagination)
├── Service Layer (app/services/media_service.py)
│   └── MediaService with business logic
├── Storage Layer (app/core/storage.py)
│   └── StorageManager for file operations
├── Processing Layer (app/core/file_processing.py)
│   ├── File validation
│   ├── Image processing & optimization
│   └── Thumbnail generation
└── API Layer (app/api/v1/endpoints/media.py)
    └── 8 RESTful endpoints
```

Technology Stack

- **FastAPI**: Web framework for API endpoints
- **SQLAlchemy**: ORM for database operations
- **Pillow**: Image processing and optimization
- **python-magic**: File type detection
- **PostgreSQL**: Database storage
- **Local Filesystem**: File storage (cloud-ready design)

API Endpoints

All media endpoints are prefixed with `/api/v1/media`.

1. Upload Media File

POST `/api/v1/media/upload`

Upload a media file with optional content association.

Authentication: Required
Request: Multipart form data
- `file`: File to upload (required)
- `media_type`: Type (image/video/audio/document) (required)
- `content_id`: Optional content UUID to associate with
- `metadata`: Optional JSON metadata

Response: `MediaResponse` with file URL

Features:
- File type validation
- Size limit enforcement
- Automatic image processing
- Thumbnail generation for images
- Metadata extraction

Example:

```
curl -X POST "http://localhost:8002/api/v1/media/upload" \
  -H "Authorization: Bearer {token}" \
  -F "file=@image.jpg" \
  -F "media_type=image" \
  -F "content_id=123e4567-e89b-12d3-a456-426614174000"
```

---

2. Upload Media for Content

POST `/api/v1/media/content/{content_id}/upload`

Upload media and automatically associate with specific content.

Authentication: Required
Path Parameters:
- `content_id`: UUID of content to associate with

Request: Multipart form data
- `file`: File to upload
- `media_type`: Type of media
- `metadata`: Optional JSON metadata

Response: `MediaResponse`

Example:

```
curl -X POST "http://localhost:8002/api/v1/media/content/{content_id}/upload" \
  -H "Authorization: Bearer {token}" \
  -F "file=@video.mp4" \
  -F "media_type=video"
```

---

3. Get Media by ID

GET `/api/v1/media/{media_id}`

Retrieve media metadata by ID.

Authentication: Public (no auth required)
Path Parameters:
- `media_id` : UUID of media

Response: `MediaResponse` with metadata

Example:

```
curl "http://localhost:8002/api/v1/media/{media_id}"
```

---

4. Download Media File

GET `/api/v1/media/{media_id}/download`

Download the actual media file.

Authentication: Public
Path Parameters:
- `media_id` : UUID of media

Response: File with proper content-type headers

Features:
- Proper content-type headers
- Original filename preservation
- Streaming for large files

Example:

```
curl "http://localhost:8002/api/v1/media/{media_id}/download" -O
```

---

5. Get Content Media

GET `/api/v1/media/content/{content_id}/media`

Get all media files associated with specific content.

Authentication: Public
Path Parameters:
- `content_id` : UUID of content

Query Parameters:
- `media_type` : Optional filter by type (image/video/audio/document)

Response: `List[MediaResponse]`

Example:

```
curl "http://localhost:8002/api/v1/media/content/{content_id}/media?media_type=image"
```

---

6. List All Media

GET `/api/v1/media/`

List all media with pagination and filters.

Authentication: Required
Query Parameters:
- `media_type` : Filter by type
- `uploaded_by` : Filter by uploader UUID
- `content_id` : Filter by content UUID
- `page` : Page number (default: 1)
- `page_size` : Items per page (default: 20, max: 100)

Response: `MediaList` with pagination metadata

Example:

```
curl "http://localhost:8002/api/v1/media/?media_type=image&page=1&page_size=20" \
  -H "Authorization: Bearer {token}"
```

---

7. Update Media Metadata

PUT `/api/v1/media/{media_id}`

Update media metadata (filename, metadata only, not the file itself).

Authentication: Required (only uploader can update)
Path Parameters:
- `media_id` : UUID of media

Request Body: `MediaUpdate` schema

```
{
  "filename": "new-filename.jpg",
  "metadata": {"location": "Uganda", "photographer": "John Doe"}
}
```

Response: `MediaResponse` with updated data

Authorization: Only the user who uploaded the media can update it.

Example:

```
curl -X PUT "http://localhost:8002/api/v1/media/{media_id}" \
  -H "Authorization: Bearer {token}" \
  -H "Content-Type: application/json" \
  -d '{"filename": "updated-name.jpg"}'
```

---

8. Delete Media

DELETE `/api/v1/media/{media_id}`

Delete media file and database record.

Authentication: Required (only uploader can delete)
Path Parameters:
- `media_id`: UUID of media

Response: Success message

Features:
- Deletes file from storage
- Deletes thumbnail if exists
- Deletes database record
- Atomic operation

Authorization: Only the user who uploaded the media can delete it.

Example:

```
curl -X DELETE "http://localhost:8002/api/v1/media/{media_id}" \
  -H "Authorization: Bearer {token}"
```

Core Features

File Validation

Supported File Types:

1. **Images** (max 10 MB)
   - JPEG/JPG
   - PNG
   - GIF
   - WebP
   - SVG

2. **Videos** (max 100 MB)
   - MP4
   - MPEG
   - QuickTime
   - AVI
   - WebM

3. **Audio** (max 50 MB)
   - MP3/MPEG
   - WAV
   - OGG
   - AAC
   - WebM

4. **Documents** (max 20 MB)
   - PDF
   - Word (DOC, DOCX)
   - Excel (XLS, XLSX)
   - Text

Validation Process:
1. MIME type detection using python-magic
2. File extension validation
3. File size checking
4. Content type verification

Image Processing

Automatic Processing for Images:

1. **Format Normalization**
   - Converts RGBA to RGB (white background)
   - Handles various color modes
   - Preserves original format when possible

2. **Size Optimization**
   - Max dimensions: 2048x2048 pixels
   - Maintains aspect ratio
   - Uses LANCZOS resampling for quality

3. **Compression**
   - JPEG quality: 85%
   - Optimize flag enabled
   - Reduces file size without visible quality loss

4. **Metadata Extraction**
   - Image dimensions (width, height)
   - Format and color mode
   - EXIF data (if available)

5. **Thumbnail Generation**
   - Size: 300x300 pixels
   - Maintains aspect ratio
   - Stored alongside original
   - Automatically generated for all images

Storage Management

Storage Architecture:

```
/home/ubuntu/content_service/uploads/
├── 2024/
│   ├── 12/
│   │   ├── filename_abc123.jpg
│   │   ├── filename_abc123_thumb.jpg
│   │   └── document_def456.pdf
│   └── ...
└── ...
```

Features:
- Hierarchical structure (year/month)
- Unique filename generation (UUID-based)
- Collision prevention
- Automatic directory creation
- Clean deletion (removes empty directories)

File Naming:
- Format: `{sanitized_name}_{unique_id}.{ext}`
- Example: `mission_photo_a1b2c3d4e5f6.jpg`
- Preserves original extension
- Limits filename length (50 chars + ID)

Cloud-Ready Design:
- Abstracted storage interface
- Easy to extend for S3, Azure, GCS
- URL generation separated from storage
- Storage path independent of serving URL

Database Schema

Media Table

```
CREATE TABLE media (
    id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    content_id UUID REFERENCES content(id) ON DELETE SET NULL,
    media_type VARCHAR(20) NOT NULL,
    filename VARCHAR(500) NOT NULL,
    url VARCHAR(1000) NOT NULL,
    storage_path VARCHAR(1000),
    file_size INTEGER,
    mime_type VARCHAR(100),
    width INTEGER,
    height INTEGER,
    duration INTEGER,
    metadata JSONB DEFAULT '{}',
    uploaded_by UUID NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

-- Indexes
CREATE INDEX idx_media_type_content ON media(media_type, content_id);
CREATE INDEX idx_uploaded_by_created ON media(uploaded_by, created_at);
CREATE INDEX idx_media_content_id ON media(content_id);
CREATE INDEX idx_media_url ON media(url);
```

Field Descriptions:
- `id` : Unique media identifier (UUID)
- `content_id` : Optional link to content (nullable)
- `media_type` : Enum (image/video/audio/document)
- `filename` : Original filename for download
- `url` : Public URL to access file
- `storage_path` : Internal storage path
- `file_size` : Size in bytes
- `mime_type` : Content type
- `width` , `height` : Dimensions (images/videos)
- `duration` : Length in seconds (audio/video)
- `metadata` : Custom JSON metadata
- `uploaded_by` : User who uploaded (from Auth Service)
- `created_at` , `updated_at` : Timestamps

Service Layer

MediaService Class

Located in `app/services/media_service.py`

Key Methods:

1. **`upload_media()`**
   - Validates file
   - Saves to storage

   - Processes images
   - Generates thumbnails
   - Extracts metadata
   - Creates database record
   - Returns Media object

2. `get_media(media_id)`
   - Retrieves media by ID
   - Includes content relationship
   - Returns None if not found

3. `get_media_for_content(content_id, media_type?)`
   - Gets all media for content
   - Optional type filtering
   - Ordered by creation date

4. `list_media(filters, pagination)`
   - Lists with filters
   - Supports pagination
   - Returns (list, total_count)

5. `update_media(media_id, update_data, user_id)`
   - Updates metadata only
   - Checks permissions
   - Returns updated media

6. `delete_media(media_id, user_id)`
   - Checks permissions
   - Deletes file from storage
   - Deletes thumbnail
   - Deletes database record
   - Returns boolean success

Error Handling:
- File validation errors (400)
- Permission errors (403)
- Not found errors (404)
- Processing errors (500)
- Automatic cleanup on failure

Security & Permissions

Authentication

Endpoints requiring authentication:
- POST `/upload` - Create media
- POST `/content/{id}/upload` - Create media for content
- GET `/` - List all media
- PUT `/{id}` - Update media
- DELETE `/{id}` - Delete media

Public endpoints:
- GET `/{id}` - Get media metadata
- GET `/{id}/download` - Download file
- GET `/content/{id}/media` - Get content media

Authorization

Permission Rules:
1. **Upload**: Any authenticated user
2. **View/Download**: Public access
3. **Update**: Only uploader
4. **Delete**: Only uploader

Implementation:
- JWT token validation via Auth Service
- User ID extraction from token
- Ownership verification in service layer
- 403 Forbidden for unauthorized actions

File Security

Validation:
- MIME type checking
- File extension verification
- Size limit enforcement
- Content inspection

Storage:
- Unique filenames prevent collisions
- Directory traversal prevention
- Sanitized filenames (alphanumeric + -_)
- Isolated storage directory

Serving:
- Proper content-type headers
- Original filename in download
- No directory listing
- Path validation

Configuration

Environment Variables

Add to `.env` file:

```
# Media & Storage
UPLOAD_DIR=/home/ubuntu/content_service/uploads
BASE_URL=http://localhost:8002
MAX_UPLOAD_SIZE=104857600  # 100 MB in bytes
```

Settings Class

Updated `app/core/config.py`:

```
class Settings(BaseSettings):
    # ... existing settings ...

    # Media & Storage
    UPLOAD_DIR: str = "/home/ubuntu/content_service/uploads"
    BASE_URL: str = "http://localhost:8002"
    MAX_UPLOAD_SIZE: int = 100 * 1024 * 1024  # 100 MB
```

File Size Limits

Defined in `app/core/file_processing.py`:

```
MAX_IMAGE_SIZE = 10 * 1024 * 1024      # 10 MB
MAX_VIDEO_SIZE = 100 * 1024 * 1024     # 100 MB
MAX_AUDIO_SIZE = 50 * 1024 * 1024      # 50 MB
MAX_DOCUMENT_SIZE = 20 * 1024 * 1024   # 20 MB
```

Image Processing Settings

```
MAX_IMAGE_WIDTH = 2048
MAX_IMAGE_HEIGHT = 2048
THUMBNAIL_SIZE = (300, 300)
IMAGE_QUALITY = 85
```

Testing

Import Validation

All modules import successfully:

```
✓ Storage module imported successfully
✓ File processing module imported successfully
✓ Media service module imported successfully
✓ Media endpoints module imported successfully
✓ Media schemas imported successfully
✓ Media model imported successfully
```

API Routes Registered

9 media routes successfully registered:

1. `/media/upload` - Upload endpoint
2. `/media/content/{content_id}/upload` - Content-specific upload
3. `/media/{media_id}` - Get by ID
4. `/media/{media_id}/download` - Download file
5. `/media/content/{content_id}/media` - Get content media
6. `/media/` - List all media
7. `/media/{media_id}` - Update media (PUT)

8. `/media/{media_id}` - Delete media (DELETE)
9. `/media/files/{file_path:path}` - Serve files (internal)

Manual Testing Checklist

- [ ] Upload image file
- [ ] Upload video file
- [ ] Upload audio file
- [ ] Upload document file
- [ ] Verify image processing
- [ ] Verify thumbnail generation
- [ ] Download file
- [ ] List media with filters
- [ ] Update media metadata
- [ ] Delete media
- [ ] Test permission restrictions
- [ ] Test file validation errors
- [ ] Test size limit enforcement

Dependencies

New Dependencies Added

```
Pillow==10.4.0          # Image processing
python-magic==0.4.27    # File type detection
```

Existing Dependencies Used

```
fastapi==0.108.0         # Web framework
sqlalchemy==2.0.25       # ORM
asyncpg==0.29.0          # PostgreSQL async driver
python-multipart==0.0.6  # File upload support
```

File Structure

New Files Created

```
app/
├── services/
│   └── media_service.py         # Media business logic
├── core/
│   ├── storage.py               # File storage management
│   └── file_processing.py       # File validation & processing
└── api/
    └── v1/
        └── endpoints/
            └── media.py         # Media API endpoints

uploads/
└── .gitkeep                     # Preserve directory in git
```

Modified Files

```
app/
├── core/
│   └── config.py                # Added media settings
└── api/
    └── v1/
        └── api.py               # Registered media router

requirements.txt                  # Added Pillow, python-magic
.gitignore                       # Excluded uploads/
```

Usage Examples

Python Client Example

```python
import httpx
import asyncio

async def upload_media():
    async with httpx.AsyncClient() as client:
        # Upload image
        with open("photo.jpg", "rb") as f:
            response = await client.post(
                "http://localhost:8002/api/v1/media/upload",
                headers={"Authorization": f"Bearer {token}"},
                files={"file": ("photo.jpg", f, "image/jpeg")},
                data={
                    "media_type": "image",
                    "content_id": "123e4567-e89b-12d3-a456-426614174000"
                }
            )

        media = response.json()
        print(f"Uploaded: {media['url']}")
        return media["id"]

async def download_media(media_id):
    async with httpx.AsyncClient() as client:
        response = await client.get(
            f"http://localhost:8002/api/v1/media/{media_id}/download"
        )

        with open("downloaded.jpg", "wb") as f:
            f.write(response.content)

# Run
media_id = asyncio.run(upload_media())
asyncio.run(download_media(media_id))
```

JavaScript/TypeScript Example

```
// Upload media
async function uploadMedia(file: File, contentId?: string) {
  const formData = new FormData();
  formData.append('file', file);
  formData.append('media_type', 'image');
  if (contentId) {
    formData.append('content_id', contentId);
  }

  const response = await fetch('http://localhost:8002/api/v1/media/upload', {
    method: 'POST',
    headers: {
      'Authorization': `Bearer ${token}`
    },
    body: formData
  });

  return await response.json();
}

// Get content media
async function getContentMedia(contentId: string) {
  const response = await fetch(
    `http://localhost:8002/api/v1/media/content/${contentId}/media`
  );

  return await response.json();
}
```

Integration with Content System

Content-Media Relationship

Database Relationship:
- Content has many Media (one-to-many)
- Media can be associated with Content (optional)
- Cascade deletion: SET NULL (keeps media when content deleted)

Usage Patterns:

1. **Attach media to existing content:**
   `python`
   ```
      # Upload with content_id
      POST /api/v1/media/upload
      {
        "file": <file>,
        "media_type": "image",
        "content_id": "<content_uuid>"
      }
   ```

2. **Get all media for content:**
   `python`
   ```
      GET /api/v1/media/content/<content_id>/media
   ```

3. **Upload directly to content:**
```
python
  POST /api/v1/media/content/<content_id>/upload
  {
    "file": <file>,
    "media_type": "image"
  }
```

Content Service Integration

Media appears in Content responses:
- `ContentWithMedia` schema includes media list
- Media automatically loaded with content
- Thumbnail URLs included for quick preview

Performance Considerations

Async Operations

All I/O operations are async:
- File reading/writing
- Database queries
- Image processing (could be optimized further)

Chunked File Uploads

- Files read in 1MB chunks
- Prevents memory overflow
- Supports large file uploads

Thumbnail Generation

- Happens during upload
- Stored alongside original
- Pre-generated for fast loading

Database Indexes

Optimized queries with indexes on:
- `media_type, content_id` (compound)
- `uploaded_by, created_at` (compound)
- `content_id` (foreign key)
- `url` (for lookups)

Potential Optimizations

1. **Background Processing:**
   - Move image processing to background task
   - Use Celery or similar
   - Return immediately after upload

2. **CDN Integration:**
   - Serve files from CDN

   - Reduce server load
   - Improve global performance

3. **Caching:**
   - Cache media metadata in Redis
   - Reduce database queries
   - TTL-based invalidation

4. **Storage Optimization:**
   - Implement cloud storage (S3, Azure)
   - Use object storage for scalability
   - Enable automatic backups

Migration Notes

Database Migration

Run migrations to create media table:

```
cd /home/ubuntu/content_service

# Generate migration
alembic revision --autogenerate -m "Add media table"

# Apply migration
alembic upgrade head
```

Initial Setup

```
# 1. Install dependencies
pip install -r requirements.txt

# 2. Create uploads directory
mkdir -p uploads

# 3. Set environment variables
export UPLOAD_DIR=/home/ubuntu/content_service/uploads
export BASE_URL=http://localhost:8002

# 4. Run migrations
alembic upgrade head

# 5. Start server
uvicorn app.main:app --host 0.0.0.0 --port 8002
```

Error Handling

Common Error Codes

400 Bad Request:
- Invalid file type

- File too large
- Invalid content_id format
- Invalid metadata JSON

403 Forbidden:
- User not authorized to update/delete
- Insufficient permissions

404 Not Found:
- Media not found
- File not found in storage
- Content not found

500 Internal Server Error:
- File processing failed
- Storage error
- Database error

Error Response Format

```
{
  "detail": "Error message describing the issue"
}
```

Client Handling

```python
try:
    response = await client.post("/api/v1/media/upload", ...)
    response.raise_for_status()
except httpx.HTTPStatusError as e:
    if e.response.status_code == 400:
        print(f"Invalid request: {e.response.json()['detail']}")
    elif e.response.status_code == 403:
        print("Permission denied")
    elif e.response.status_code == 404:
        print("Resource not found")
    else:
        print(f"Server error: {e.response.status_code}")
```

Future Enhancements

Short-term (Next Sprint)

1. **Video Processing:**
   - Extract thumbnails from videos
   - Get video duration and metadata
   - Support for video formats

2. **Audio Processing:**
   - Extract audio duration
   - Support for audio metadata
   - Waveform generation

3. **Batch Upload:**
   - Upload multiple files at once
   - Progress tracking
   - Partial success handling

4. **Media Gallery:**
   - Grid view of images
   - Lightbox preview
   - Sorting and filtering

Medium-term

1. **Cloud Storage:**
   - S3 integration
   - Azure Blob Storage
   - Google Cloud Storage
   - Configurable storage backend

2. **Advanced Image Processing:**
   - Multiple thumbnail sizes
   - Image cropping/editing
   - Watermarking
   - Format conversion

3. **CDN Integration:**
   - CloudFlare integration
   - Automatic cache purging
   - Edge caching

4. **Media Transformations:**
   - On-the-fly resizing
   - Format conversion
   - Quality adjustment
   - URL-based transformations

Long-term

1. **AI Features:**
   - Auto-tagging images
   - Object detection
   - Face recognition
   - Content moderation

2. **Advanced Search:**
   - Full-text search in documents
   - Image similarity search
   - Reverse image search

3. **Analytics:**
   - Download tracking
   - Usage statistics
   - Popular media reports

4. **Collaboration:**
   - Media comments
   - Approval workflows
   - Version history
   - Media collections

API Documentation

Interactive Docs

Once the server is running, access interactive API documentation:

- **Swagger UI:** http://localhost:8002/docs
- **ReDoc:** http://localhost:8002/redoc

OpenAPI Specification

Download OpenAPI JSON:

```
curl http://localhost:8002/openapi.json > media-api-spec.json
```

Support & Troubleshooting

Common Issues

1. File upload fails with 413 (Payload Too Large)
- Check MAX_UPLOAD_SIZE in config
- Adjust nginx/proxy settings
- Verify file size limits

2. Image processing fails
- Ensure Pillow is installed correctly
- Check image format support
- Verify file is not corrupted

3. Files not accessible
- Verify UPLOAD_DIR permissions
- Check BASE_URL configuration
- Ensure storage path is correct

4. Thumbnails not generating
- Check Pillow installation
- Verify write permissions
- Check disk space

Debug Mode

Enable debug logging:

```
# In config.py
LOG_LEVEL: str = "DEBUG"
DEBUG: bool = True
```

View detailed logs:

```
tail -f logs/content_service.log
```

Deployment Checklist

Pre-deployment

- [ ] Run database migrations
- [ ] Create uploads directory
- [ ] Set environment variables
- [ ] Install system dependencies (libmagic)
- [ ] Configure file size limits
- [ ] Set up backup for uploads directory

Production Settings

```
# .env production settings
ENVIRONMENT=production
DEBUG=False
LOG_LEVEL=INFO

# Use absolute paths
UPLOAD_DIR=/var/www/content_service/uploads

# Production URL
BASE_URL=https://api.engadi.org

# Increase limits for production
MAX_UPLOAD_SIZE=209715200  # 200 MB
```

Monitoring

- Set up file storage monitoring
- Track upload success/failure rates
- Monitor disk space usage
- Log slow image processing
- Alert on errors

Version Control

Git Status

```
# New files
app/services/media_service.py
app/core/storage.py
app/core/file_processing.py
app/api/v1/endpoints/media.py
uploads/.gitkeep
MEDIA_API_IMPLEMENTATION_SUMMARY.md

# Modified files
app/core/config.py
app/api/v1/api.py
requirements.txt
.gitignore
```

Commit Message

```
feat: Implement Media Handling System

- Add media upload/download endpoints (8 endpoints)
- Implement image processing with Pillow
- Add thumbnail generation for images
- Create storage management system
- Add file validation and security checks
- Support images, videos, audio, documents
- Implement permission-based access control
- Add comprehensive error handling

Components:
- MediaService: Business logic layer
- StorageManager: File storage operations
- File processing: Validation and optimization
- Media API: 8 RESTful endpoints

Dependencies:
- Pillow 10.4.0: Image processing
- python-magic 0.4.27: File type detection

Closes #XXX
```

Summary

Implementation Status

✅ **Completed:**
- Media Service Layer
- Storage Utilities
- File Processing Utilities
- 8 API Endpoints
- Image Processing
- Thumbnail Generation

- File Validation
- Security & Permissions
- Error Handling
- Configuration
- Documentation

Statistics

- **Files Created:** 5 new files
- **Files Modified:** 4 existing files
- **Lines of Code:** ~1,200 LOC
- **API Endpoints:** 8 documented + 1 internal
- **Supported File Types:** 20+ formats
- **Media Types:** 4 (image, video, audio, document)
- **Test Coverage:** Import validation completed

Key Achievements

1. **Comprehensive File Support:** Images, videos, audio, documents
2. **Automatic Processing:** Image optimization and thumbnail generation
3. **Secure Upload:** File validation and permission checking
4. **Cloud-Ready Design:** Abstracted storage layer
5. **RESTful API:** Follows best practices
6. **Type-Safe:** Full type hints with Pydantic
7. **Async Operations:** Scalable and performant
8. **Well-Documented:** Complete API documentation

Next Steps

1. **Database Migration:**
   `bash`
   ```
   alembic revision --autogenerate -m "Add media table"
   alembic upgrade head
   ```

2. **Install Dependencies:**
   `bash`
   ```
   pip install -r requirements.txt
   ```

3. **Start Service:**
   `bash`
   ```
   uvicorn app.main:app --reload
   ```

4. **Test Endpoints:**
   - Access Swagger UI at http://localhost:8002/docs
   - Test file upload
   - Verify image processing
   - Check thumbnail generation

5. **Integration Testing:**
   - Test with frontend

- Verify content association
- Test permission system
- Load testing

6. **Monitoring Setup:**
   - Configure logging
   - Set up alerts
   - Monitor storage usage
   - Track performance metrics

Implementation Date: December 22, 2024
Author: DeepAgent (Abacus.AI)
Version: 1.0.0
Status: ✅ Complete and Ready for Testing