Content Service Generation Summary

Overview

The **Content Service** has been successfully generated and configured as part of the Mission Engadi microservices architecture. This service handles missions content management with multi-language support, enabling the organization to share stories, updates, testimonials, prayer requests, and blog posts across multiple languages.

Service Port: 8002
Generated: December 22, 2025
Template Used: mission-engadi-service-template
Base Framework: FastAPI + PostgreSQL + SQLAlchemy (Async)

Project Structure

```
content_service/
├── app/
│   ├── api/v1/              # API routes (versioned)
│   ├── core/                # Configuration, security, logging
│   ├── db/                  # Database session and base classes
│   ├── dependencies/        # Dependency injection
│   ├── models/              # SQLAlchemy models
│   │   ├── content.py        # ✓ Content model implemented
│   │   ├── translation.py    # ✓ Translation model implemented
│   │   └── media.py          # ✓ Media model implemented
│   ├── schemas/             # Pydantic validation schemas
│   │   ├── content.py        # ✓ Content schemas implemented
│   │   ├── translation.py    # ✓ Translation schemas implemented
│   │   └── media.py          # ✓ Media schemas implemented
│   ├── services/            # Business logic layer
│   └── main.py              # FastAPI application entry point
├── migrations/
│   └── versions/
│       └── 2025_12_22_...    # ✓ Initial migration created
├── tests/                   # Unit and integration tests
├── .env.example             # ✓ Environment configuration template
├── alembic.ini              # ✓ Database migration configuration
├── docker-compose.yml       # Docker setup for local development
├── Dockerfile               # Container image definition
├── requirements.txt         # Python dependencies
└── README.md                # Project documentation
```

Database Models Implemented

1. Content Model (`app/models/content.py`)

The core model for managing missions content.

Fields:
- `id` (UUID) - Primary key
- `title` (String, indexed) - Content title
- `slug` (String, unique, indexed) - URL-friendly identifier
- `body` (Text) - Main content body (supports markdown)
- `content_type` (Enum) - Type: story, update, testimonial, prayer_request, blog_post
- `status` (Enum) - Status: draft, review, published, archived
- `author_id` (UUID, indexed) - Foreign key to Auth Service users
- `language` (String, indexed) - Default ‘en’ (English)
- `featured_image_url` (String, nullable) - Featured image URL
- `tags` (Array[String]) - Tags for categorization
- `meta` (JSONB) - Additional metadata
- `published_at` (DateTime, indexed, nullable) - Publication timestamp
- `created_at`, `updated_at` (DateTime) - Automatic timestamps

Relationships:
- `translations` → One-to-Many with Translation (cascade delete)
- `media` → One-to-Many with Media (cascade delete)

Indexes:
- Composite: content_type + status
- Composite: language + status
- Composite: author_id + status
- Descending: published_at

---

2. Translation Model (`app/models/translation.py`)

Manages multi-language translations of content.

Fields:
- `id` (UUID) - Primary key
- `content_id` (UUID, FK, indexed) - References content.id (CASCADE on delete)
- `language` (String, indexed) - Target language code (en, es, fr, pt-br)
- `translated_title` (String, indexed) - Translated title
- `translated_body` (Text) - Translated content body
- `translated_slug` (String, indexed) - Translated URL slug
- `translator_id` (UUID, indexed, nullable) - Foreign key to Auth Service users
- `translation_status` (Enum) - Status: pending, in_progress, completed, reviewed
- `created_at`, `updated_at` (DateTime) - Automatic timestamps

Relationships:
- `content` → Many-to-One with Content

Indexes:
- Unique composite: content_id + language (one translation per language per content)
- Composite: language + translation_status

---

3. Media Model (`app/models/media.py`)

Handles media files associated with content.

Fields:
- `id` (UUID) - Primary key
- `content_id` (UUID, FK, indexed, nullable) - References content.id (SET NULL on delete)
- `media_type` (String, indexed) - Type: image, video, audio, document
- `filename` (String) - Original filename
- `url` (String, indexed) - Public URL to media
- `storage_path` (String, nullable) - Storage system path (e.g., S3 key)
- `file_size` (Integer, nullable) - File size in bytes
- `mime_type` (String, nullable) - MIME type
- `width` (Integer, nullable) - Width in pixels (images/videos)
- `height` (Integer, nullable) - Height in pixels (images/videos)
- `duration` (Integer, nullable) - Duration in seconds (audio/video)
- `meta` (JSONB) - Additional metadata (EXIF, location, etc.)
- `uploaded_by` (UUID, indexed) - Foreign key to Auth Service users
- `created_at` (DateTime) - Upload timestamp

Relationships:
- `content` → Many-to-One with Content (optional)

Indexes:
- Composite: media_type + content_id
- Composite: uploaded_by + created_at

Pydantic Schemas Implemented

Content Schemas (`app/schemas/content.py`)

- `ContentBase` - Base fields for all content operations
- `ContentCreate` - Schema for creating new content
- `ContentUpdate` - Schema for partial updates (all optional fields)
- `ContentInDB` - Internal database representation
- `ContentResponse` - API response with all fields
- `ContentWithTranslations` - Response including translations
- `ContentWithMedia` - Response including media
- `ContentFull` - Complete response with translations + media
- `ContentList` - Paginated list response

Translation Schemas (`app/schemas/translation.py`)

- `TranslationBase` - Base translation fields
- `TranslationCreate` - Create new translation
- `TranslationUpdate` - Partial update schema
- `TranslationInDB` - Database representation
- `TranslationResponse` - API response
- `TranslationList` - Paginated list response

Media Schemas (`app/schemas/media.py`)

- `MediaBase` - Base media fields
- `MediaCreate` - Create new media

- `MediaUpdate` - Partial update schema
- `MediaInDB` - Database representation
- `MediaResponse` - API response
- `MediaList` - Paginated list response

Database Migration

Migration File: `migrations/versions/2025_12_22_0813_bfc6d174900f_initial_migration_content_translation_.py`

Status: ✓ Generated and ready for deployment

Contains:
- Creation of PostgreSQL ENUMs (content_type, content_status, translation_status)
- Content table with all fields and indexes
- Translations table with foreign key to content
- Media table with foreign key to content
- All composite indexes for optimized queries
- Foreign key constraints with CASCADE/SET NULL policies

To Apply:

```
cd /home/ubuntu/content_service
alembic upgrade head
```

To Rollback:

```
alembic downgrade -1
```

Configuration

Environment Variables (`.env`)

Key configurations:
- **PROJECT_NAME:** "Content Service"
- **PORT:** 8002
- **DATABASE_URL:** postgresql+asyncpg://postgres:postgres@localhost:5432/content_service_db
- **AUTH_SERVICE_URL:** http://localhost:8001
- **CORS_ORIGINS:** Configured for local development
- **REDIS_URL:** redis://localhost:6379/0 (for caching)
- **KAFKA_BOOTSTRAP_SERVERS:** localhost:9092 (for events)

Dependencies (`requirements.txt`)

- FastAPI 0.108.0 - Web framework
- SQLAlchemy 2.0.25 (asyncio) - ORM
- Alembic 1.13.1 - Database migrations

- Asyncpg 0.29.0 - PostgreSQL async driver
- Pydantic 2.5.3 - Data validation
- Python-jose 3.3.0 - JWT authentication
- Redis 5.0.1 - Caching
- aiokafka 0.10.0 - Event streaming

Verification Results

✓ Models Import Successfully

- Content, Translation, Media
- All enums (ContentType, ContentStatus, TranslationStatus, MediaType)

✓ Schemas Import Successfully

- All create, update, and response schemas
- Forward references working correctly

✓ Settings Loaded

- Configuration from .env file
- Database URL configured
- CORS origins parsed correctly

✓ Database Base Class

- SQLAlchemy DeclarativeBase working
- Async support enabled

Next Steps

Immediate Tasks

1. **Apply Database Migration**
   `bash`
   ```
      cd /home/ubuntu/content_service
      docker-compose up -d  # Start PostgreSQL
      alembic upgrade head  # Apply migrations
   ```

2. **Implement API Endpoints**
   - Create `/api/v1/endpoints/content.py` for CRUD operations
   - Create `/api/v1/endpoints/translations.py` for translation management
   - Create `/api/v1/endpoints/media.py` for media uploads
   - Register endpoints in `/api/v1/api.py`

3. **Implement Service Layer**
   - Create `app/services/content_service.py` for business logic
   - Create `app/services/translation_service.py`
   - Create `app/services/media_service.py`
   - Add validation and authorization logic

4. **Add External Integrations**
   - AI translation service integration (for auto-translation)
   - Media storage service (S3/CloudFlare R2)
   - Social media APIs (for sharing)

5. **Write Tests**
   - Unit tests for models and schemas
   - Integration tests for API endpoints
   - Test fixtures for sample content

6. **Documentation**
   - Update README.md with API documentation
   - Add OpenAPI schema examples
   - Document translation workflow

Long-term Enhancements

1. **Content Workflow**
   - Draft → Review → Publish pipeline
   - Approval system for content
   - Content versioning

2. **AI Features**
   - Automatic translation suggestions
   - Content summarization
   - Tag recommendations

3. **Performance Optimization**
   - Redis caching for published content
   - CDN integration for media
   - Database query optimization

4. **Analytics**
   - Content view tracking
   - Translation coverage metrics
   - Popular content reporting

External Service Dependencies

Auth Service (Port 8001)

- User authentication
- Author and translator user data
- Required for: author_id, uploaded_by, translator_id fields

Storage Service (Future)

- Media file storage (images, videos, audio, documents)
- CDN integration
- Required for: Media model file uploads

Translation Service (Future - Optional)

- AI-powered auto-translation
- Translation quality scoring
- Required for: Automated translation workflow

Git Repository Status

Location: `/home/ubuntu/content_service`

Status:
- Git initialized
- All files staged
- Ready for initial commit

Next Git Steps:

```
cd /home/ubuntu/content_service
git add .
git commit -m "Initial commit: Content Service with core models and schemas"
git remote add origin <github-url>
git push -u origin main
```

Technology Stack Summary

Component	Technology	Version
Framework	FastAPI	0.108.0
Language	Python	3.11+
Database	PostgreSQL	15+
ORM	SQLAlchemy (async)	2.0.25
Migrations	Alembic	1.13.1
Validation	Pydantic	2.5.3
Cache	Redis	5.0.1
Events	Kafka (aiokafka)	0.10.0
Server	Uvicorn	0.25.0
Container	Docker	Latest

API Documentation

Once the service is running, access interactive API documentation at:

- **Swagger UI:** http://localhost:8002/api/v1/docs
- **ReDoc:** http://localhost:8002/api/v1/redoc
- **OpenAPI JSON:** http://localhost:8002/api/v1/openapi.json

---

Support & Contact

For questions or issues related to the Content Service:

1. Check the main README.md for setup instructions
2. Review CONTRIBUTING.md for development guidelines
3. Contact the Mission Engadi development team

Generated by: DeepAgent (Abacus.AI)
Date: December 22, 2025
Status: ✓ Ready for Development